МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ ГОРОДСКОГО ХОЗЯЙСТВА имени А. Н. БЕКЕТОВА

Е. Е. Поморцева

ПРОГРАММИРОВАНИЕ ГЕОИНФОРМАЦИОННЫХ ЗАДАЧ

УЧЕБНОЕ ПОСОБИЕ

Харьков
ХНУГХ им. А. Н. Бекетова
2017

УДК 004.42:004.451.42(075)
П55

Автор

Поморцева *Елена Евгеньевна*, канд. техн. наук, доц.

Рецензенты:

А. А. Янцевич, д-р физ.-мат. наук, профессор кафедры математических методов в экономике ХНУ им. В. Н. Каразина;

А. Г. Колгатин, д-р пед. наук, профессор кафедры информатики ХНПУ им. Г. С. Сковороды

Рекомендовано к печати на заседании
Ученого совета ХНУГХ им. А. Н. Бекетова,
протокол № 8 от 27 января 2017 г.

Поморцева Е. Е.

П55 Программирование геоинформационных задач : учеб. пособие / Е. Е. Поморцева; Харьков. нац. ун-т гор. хоз-ва им. А. Н. Бекетова. – Харьков : ХНУГХ им. А. Н. Бекетова, 2017. – 121 с.

ISBN 978-966-695-433-9

Материал пособия состоит из двух разделов, позволяющих не только изучить теоретическую сторону вопроса, но и рассмотреть основные Case-средства, используемые специалистами геоинформационного направления в ходе решения прикладных задач. Изложенный материал насыщен примерами из прикладной области, что позволяет изучить основные приемы использования интегрированной среды разработки VBA в решении задач будущей профессиональной деятельности. Достигнутый уровень компетентности позволит эффективно использовать возможности языка программирования Visual Basic for Applications при решении профессиональных задач студентами направления подготовки «Геодезия, картография и землеустройство», создаст основу для самостоятельного овладения новыми языками программирования.

УДК 004.42:004.451.42(075)

ISBN 978-966-695-433-9

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

CLSID – идентификатор класса (Class Identifier)

COM – компонентная модель объектов (Component Object Model)

DDE – динамический обмен данными

DLL – динамически подключаемая библиотека (Dynamic Link Library)

FID – поле с автонумерацией

GUID – глобальный уникальный идентификатор (Globally Unique Identifier)

IDL – язык описания интерфейса (Interface Definition Language).

RAD – Rapid Application Development

UML – унифицированный язык моделирования (Unified Modelling Language)

VB – Visual Basic

VBA – Visual Basic for Applications

АИС – система для атрибутивных данных

АИС – автоматизированная информационная система

БГД – баз геоданных

БД – банк данных

ИСР – интегрированная среда разработки (IDE – Integrated development environment)

МКО – модель компонентных объектов

ООА – объектно-ориентированный анализ

ООД – объектно-ориентированный дизайн

ООП – объектно-ориентированное проектирование

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базой данных

ЭУ – элемент управления

ВВЕДЕНИЕ

Учебное пособие подготовлено в соответствии с программой учебной дисциплины по выбору «Программирование геоинформационных задач», которая имеет профессиональное значение для подготовки бакалавров по направлению «Геодезия, картография и землеустройство».

Учебное пособие состоит из следующих подразделов:

Раздел 1. Приемы использования интегрированной среды разработки Visual Basic for Application.

- История создания и развития VBA.
- Принципы визуального программирования.
- Принципы разработки интерфейса.
- Переменные и типы данных.
- Базовые конструкции языка VBA.
- Принципы структурного программирования.
- Объектно-ориентированное программирование.
- Применение UML.
- Принципы экстремального программирования.

Раздел 2. Программирование прикладных геоинформационных задач.

- Язык VBA для программной платформы ArcGis Desktop.
- Технология COM в ArcObjects.
- Модули, объекты и классы.
- Некоторые ресурсы для разработчиков.
- Компонент карта как объект. Его методы, свойства и элементы.
- Организация доступа к картам и тематическим слоям. Алгоритмы размещения слоев в карте.
- Управление атрибутивными данными.

Цель и задачи изучения дисциплины

Целью данной дисциплины является изучение теоретических принципов и основ программирования прикладных ГИС-задач, а также концепции объектно-ориентированного программирования, использования объектных диаграмм, программирование с использованием объектов геоданных. С помощью этого пособия читатель сможет овладеть навыками работы с программным обеспечением ARCGIS Desktop.

Задачей дисциплины есть предоставление сведений о современных компьютерных технологиях для последующего их использования в работе

по выбранной специальноси, о современном состоянии специализированного программного обеспечения. В результате освоения материала читатель сможет самостоятельно использовать полученные знания в ходе освоения новых программных продуктов в практической работе.

Предмет изучения дисциплины

Предметом дисциплины является технология разработки, создания и использования программных продуктов для конечного пользователя с помощью языка программирования Visual Basic for Applications – упрощенной версии Visual Basic, для решения геоинформационных задач, задач землеустройства и кадастра.

В данном учебном пособии рассмотрены основные теоретические понятия и термины, которые помогают овладеть основными принципами объектно-ориентированного программирования проектирования, в частности в таких Case-средствах, как ArcMap и ArcCatalog.

Для улучшения усвоения изложенного материала в конце каждого подраздела учебного пособия приведены вопросы для самопроверки, что позволит закрепить полученные знания и проконтролировать их усвоение.

Профессиональные компетентности, которые формируются при изучении учебной дисциплины «Программирование геоинформационных задач».

В процессе обучения студенты получают необходимые знания во время лекционных занятий, закрепляют и углубляют их, приобретая при этом практические навыки и умения при выполнении лабораторных работ. Особое значение имеет индивидуальная работа студентов, при выполнении которой они самостоятельно создают ряд проектов в среде ArcMap для решения задач управления муниципальными процессами. В процессе работы над проектом студенты приобретают навыки работы с научно-технической литературой, учатся самостоятельно принимать решения и анализировать. В результате усвоения материала учебного пособия у студентов должны сформироваться следующие компетентности.

Проектные, связанные с использованием основ программирования в VBA, освоением основных принципов создания приложений.

Аналитические, связанные с использованием прикладных пакетов для обработки и анализа данных предметной области в намеченные сроки с использованием программирования, самостоятельного выбора и освоения новых программных продуктов.

Управленческие, связанные с применением визуального языка программирования Visual Basic для комплексной обработки геоинформационных данных, и созданием приложений для решения прикладных задач.

Принципы, лежащие в основе построения учебного пособия

В данном пособии используются принципы системности и практической направленности с использованием современного программного обеспечения от ведущих разработчиков в области геоинформационных технологий.

Принцип практической направленности предусматривает фундаментальную научную подготовку и активное практическое обучение студентов. Учебное пособие создает условия для формирования обширной теоретической базы для последующего использования этих знаний на практике.

В пособии рассматриваются англоязычные версии программных продуктов, поэтому названия элементов управления, меню, команды и соответствующие им диалоговые окна приводятся на английском языке.

Важно, что приведенные примеры имеют практическую нацеленность касательно подготовки студентов. Формирование навыков может осуществляться как под руководством преподавателя в аудитории, так и дома путем самостоятельного изучения изложенного материала.

При выборе материала учитывались ограничения, которые накладывает на учебный процесс количество отведенных кредитов и часов изучения дисциплины. В пособие включен необходимый набор тем, без которых невозможна осмысленная и эффективная работа в плане разработки приложений на VBA.

В ходе изложения материала используются **навигационные подсказки,** которые помогут ориентироваться в структуре учебного пособия:

полужирное начертание – термины используемых Case-средств;

курсивное начертание – названия, которые вводит студент;

ПРИМЕР – разъяснение, с помощью которого описано решение задачи.

Усвоение в полном объеме материала данного учебного пособия поможет развить способность к дальнейшему обучению, самостоятельному развитию и овладению объектно-ориентированными языками программирования.

Учебное пособие апробировано во время проведения лекций и лабораторных работ по дисциплине «Программирование геоинформационных задач» студентам Харьковского национального университета городского хозяйства имени А. Н. Бекетова.

РАЗДЕЛ 1 ПРИЕМЫ ИСПОЛЬЗОВАНИЯ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ VISUAL BASIC FOR APPLICATION

1.1 ИСТОРИЯ СОЗДАНИЯ И РАЗВИТИЯ VBA

Язык Basic был разработан профессорами Дартмутского колледжа Дж. Кемени и Т. Курцом в 1965 году как средство обучения и работы непрофессиональных программистов. Его назначение определено в самом названии, которое является аббревиатурой слов Beginner's All-purpose Symbolic Instruction Code (многоцелевой язык символических инструкций для начинающих) и при этом в дословном переводе означает «Базовый».

Однако парадокс заключался в том, что, будучи действительно весьма простым средством программирования, совершенно непригодным в те времена для решения серьезных задач, Basic представлял собой качественно новую технологию создания программ в режиме интерактивного диалога между разработчиком и компьютером. Вернее, представлял собой праобраз современных систем программирования. Другое дело, что решение подобной задачи на технике тех лет было возможно только за счет максимального упрощения языка программирования и использования транслятора типа «интерпретатор».

В силу этих же причин Basic в основном применялся на мини- и микро-ЭВМ, которые в 70-е годы XX в. имели оперативную память, объем которой кажется сегодня просто нереальным (4–32 тысяч байт). Резкое развитие систем на основе Basic началось с появлением в начале 80-х годов персональных компьютеров, производительность и популярность которых растет в последнее время невиданными темпами.

В начале 90-х годов XX в. Microsoft начала активную борьбу за продвижение на рынок своей новой операционной системы (ОС) Windows (вместо MS-DOS). Однако, как известно, пользователи работают не с ОС, а с программами, которые работают в ее среде, поэтому скорость смены платформы в основном определяется темпами появления соответствующих прикладных программ.

Однако смена операционных систем представляет серьезную проблему и для программистов, так как им нужно было осваивать новую технологию разработки программ. В тот момент бытующим мнением было то, что ОС Windows предъявляла более высокие требования к квалификации программиста.

В 1991 году под лозунгом «теперь и начинающие программисты могут легко создавать приложения для Windows» появилась первая версия нового инструментального средства Microsoft Visual Basic (VB). В тот момент Microsoft достаточно скромно оценивала возможности этой системы, ориентируя ее, прежде всего, на категорию начинающих и непрофессиональных программистов. Основной задачей было выпустить на рынок простой и удобный инструмент разработки в тогда еще довольно новой среде Windows, программирование в которой представляло проблему и для опытных специалистов. Тем более что конкуренция со стороны других разработчиков языков программирования под ОС Windows на тот момент была не высока (рис. 1.1).

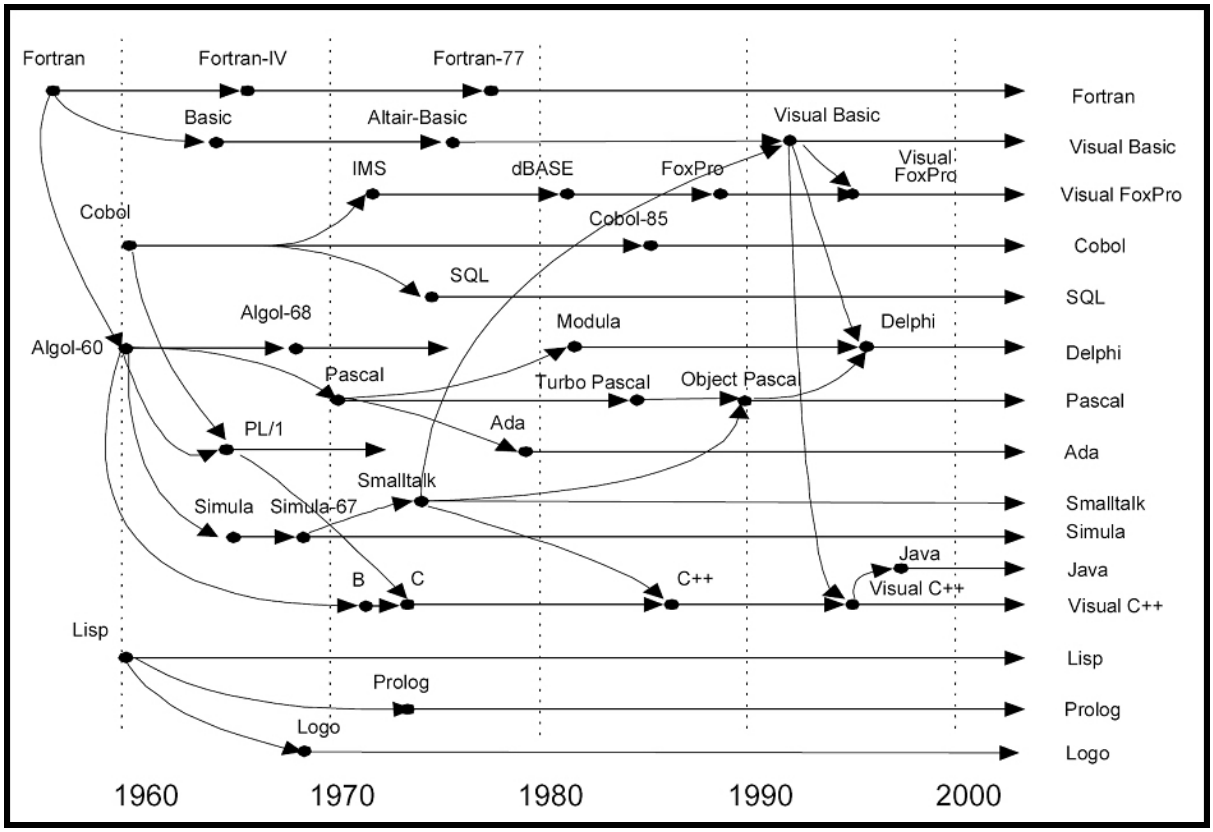


Рисунок 1.1 – Хронология развития языков программирования

Действительно, VB 1.0 в тот момент был больше похож не на рабочий инструмент, а на действующий макет будущей среды разработки. Его принципиальное новшество заключалось в реализации идей событийно-управляемого и визуального программирования в среде Windows, которые весьма радикально отличались от классических схем разработки программ.

По общему признанию VB стал родоначальником нового поколения инструментов, называемых сегодня средствами быстрой разработки программ (Rapid Application Development – RAD). Сегодня эта идеология считается привычной, но тогда она казалась совершенно необычной и создавала серьезные проблемы (в том числе чисто психологического плана) для программистов.

Тем не менее, число VB-пользователей росло, причем во многом за счет огромной популярности предшественника данного языка программирования – QuickBasic. При этом VB быстро усиливался как за счет развития среды программирования, так и за счет включения профессиональных элементов языка и проблемно-ориентированных средств. И к моменту выпуска в 1995 году VB версии 4.0 эта система была уже признанным и одним из самых распространенных инструментов создания широкого класса приложений. В начале 90-х годов наметилась отчетливая тенденция включение в приложения, предназначенные для конечного пользователя, средств внутреннего программирования, которые должны были решать задачи настройки и адаптации этих пакетов для конкретных условий их применения. В конце 1993 г. Microsoft объявила о намерении создать на основе VB новую универсальную систему программирования для прикладных программ, которая получила название Visual Basic for Applications (VBA) – VB для приложений. Вполне поятно, что реализацию этого проекта она начала с собственных офисных пакетов.

Первый вариант VBA 1.0 появился в составе MS Office 4.0, но лишь в программах Excel 4.0 и Project 6.0. В других же приложениях – Word 6.0 и Access 2.0 – были собственные варианты Basic. Более того, VBA 1.0 довольно сильно отличался (причем имея ряд существенных преимуществ) от используемой тогда универсальной системы VB 3.0. Качественный перелом наступил в конце 1996 года с выпуском MS Office 97, в котором была реализована единая среда программирования VBA 5.0, включенная в программы Word, Excel и PowerPoint. Кроме того, VBA 5.0 использовала тот же самый языковый механизм и среду разработки, что и универсальная система VB 5.0. В состав выпущенного MS Office 2000 вошла соответственно версия VBA 6.0, которая использовалась в шести программах – Word, Excel, PowerPoint, Access, Outlook, FrontPage. В результате последнее время Microsoft позиционирует свой пакет MS Office не просто как набор прикладных программ, а как комплексную платформу для создания

бизнес-приложений, решающих широкий круг специализированных задач пользователей.

Одновременно Microsoft активно продвигает VBA в качестве отраслевого стандарта для управления программируемыми приложениями, объявив о возможности его лицензирования. Сегодня уже более ста ведущих мировых фирм-разработчиков прикладных программ приобрели лицензии на него и включают VBA в состав своих программных продуктов, в том числе и американская компания ESRI – производитель геоинформационных систем.

Освоение механизма программирования VBA, реализованного в офисном приложении открывает пользователям возможность использования полученных знаний и навыков при работе с десятками и сотнями других программ, в том числе и тех, которых пока еще не существует. Начав с создания простейших макрокоманд, при желании можно в рамках одного инструментария стать профессионалом, разрабатывающим программные системы любой сложности. Буквально десять – двадцать лет назад во всем мире было не более двух миллионов программистов. Сегодня их насчитывается около десяти миллионов, из них не менее 70 процентов используют хотя бы один из инструментов VB или VBA.

Вопросы для самоконтроля

1. Кто разработал язык программирования Basic?
2. Каково назначение языка Basic и как расшифровывается его аббревиатура?
3. Когда и в силу каких причин появился Visual Basic?
4. Опишите историю появления и развития Visual Basic for Applications (VBA).
5. Входит ли VBA в программные продукты американской компании ESRI и какие задачи позволяет решать?

1.2 ПРИНЦИПЫ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ

Революционным шагом в программировании, кардинально облегчившим жизнь программистов, явилось появление визуального программирования, возникшего в Visual Basic и нашедшего воплощение в системах C++Builder, Microsoft Visual C++, Delphi фирмы Borland и других.

Визуальное программирование позволило свести проектирование пользовательского интерфейса к простым и наглядным процедурам, кото-

рые дают возможность за минуты или часы сделать то, на что ранее уходили месяцы работы.

Приложение, построенное по принципам объектной ориентации – это не последовательность каких-то операторов, не является неким жестким алгоритмом. Объектно-ориентрованная программа – это совокупность объектов и способов их взаимодействия.

Объект – это сущность, способная сохранять свое состояние и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния. Внешнее управление объектом осуществляется через обработчики событий. Эти обработчики обращаются к методам и свойствам объекта. Начальные значения данных объекта могут задаваться также в процессе проектирования установкой различных свойств. В результате выполнения методов объекта могут генерироваться новые события, воспринимаемые другими объектами программы или пользователем.

Отдельным и главным объектом при таком подходе во многих случаях можно считать пользователя программы. Он же служит основным, но не единственным, источником событий, управляющих приложением.

Средой взаимодействия объектов являются сообщения, генерируемые в результате различных событий. События наступают, прежде всего, вследствие действий пользователя – перемещения курсора мыши, нажатия кнопок мыши или клавиш клавиатуры. Но события могут наступать и в результате работы самих объектов. В каждом объекте определено множество событий, на которые он может реагировать. В конкретных экземплярах объекта могут быть определены обработчики каких-то из этих событий, которые и определяют реакцию данного экземпляра объекта. К написанию этих обработчиков, часто весьма простых, и сводится основное программирование при разработке графического интерфейса пользователя с помощью Visual Basic.

Создание обработчиков событий в Visual Basic

Вы работаете в интегрированной среде разработки (ИСР или Integrated development environment – IDE). Среда предоставляет вам формы (в приложении их может быть несколько), на которых размещаются компоненты. Обычно это оконные формы, хотя они могут быть невидимыми. На форму с помощью мыши переносятся и размещаются пиктограммы элементов управления (рис. 1.2).

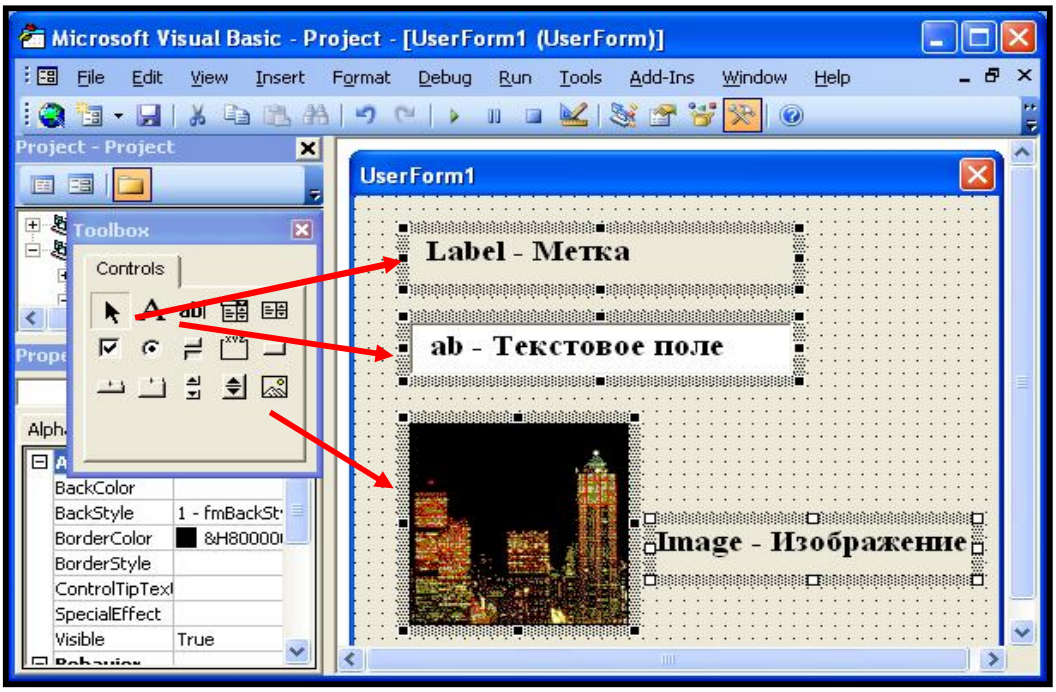


Рисунок 1.2 – Создание оконной формы

С помощью простых манипуляций мышью можно изменять размеры и расположение этих компонентов. При этом все время в процессе проектирования виден результат – изображение формы и расположенных на ней компонентов. Не нужно многократно запускать приложение и выбирать наиболее удачные размеры окна и компонентов для просмотра формы. Результаты проектирования видны, даже не компилируя программу (транслируя програму, составленную на языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду) немедленно после выполнения какой-то операции с помощью мыши.

Однако достоинства визуального программирования не сводятся к этому. Самое главное заключается в том, что во время проектирования формы и размещения на ней компонентов VB автоматически формирует коды программы, включая в нее соответствующие фрагменты, описывающие данный компонент. Затем в соответствующих диалоговых окнах пользователь может изменить заданные по умолчанию значения каких-то свойств этих компонентов, а при необходимости написать обработчики каких-то событий.

Таким образом, проектирование сводится фактически к размещению компонентов на форме, заданию некоторых их свойств и написанию, при необходимости, обработчиков событий.

Компоненты могут быть визуальные, видимые при работе приложения, и невизуальные, выполняющие те или иные служебные функции. Визуальные компоненты сразу видны на экране в процессе проектирования в таком же виде, в каком их увидит пользователь во время выполнения приложения. Это позволяет очень легко выбрать место их расположения и их дизайн – форму, размер, оформление, текст, цвет. Невизуальные компоненты видны на форме в процессе проектирования в виде пиктограмм, но пользователю во время выполнения программы они не видны, хотя и выполняют для него за кадром весьма полезную работу.

Вопросы для самоконтроля

1. В чем основное отличие визуального программирования?
2. Опишите составляющие объектно-ориентированной программы.
3. Поясните что такое событие, сообщение с точки зрения пользователя. Приведите примеры.
4. В чем заключается смысл компиляции программы?
5. Опишите назначение визуальных и невизуальных компонентов (объектов) формы.

1.3 ПРИНЦИПЫ РАЗРАБОТКИ ИНТЕРФЕЙСА

Основное достоинство Windows-приложений – их стандартный вид. Если пользователь научился работать в одном из них, то можно считать, что он без труда освоит любое. Все приложения Windows используют одни и те же соглашения и принципы работы (рис. 1.3). Существует несколько общих принципов разработки приложений под Windows. Следуя им, разработчик получает ряд существенных преимуществ. Во-первых, разработанное приложение выглядит профессионально. Во-вторых, оно легко осваивается конечным пользователем и согласуется с другими приложениями, и в-третьих, имеет современный дизайн.

Интерфейс – это внешняя оболочка приложения вместе с программами управления и другими скрытыми от пользователя механизмами управления, которые дают возможность работать с документами и данными.

Главная цель любого приложения обеспечить удобство и эффективность работы с информацией: документами, базами данных, графикой или изображениями. Поэтому интерфейс является важнейшей частью любого приложения.

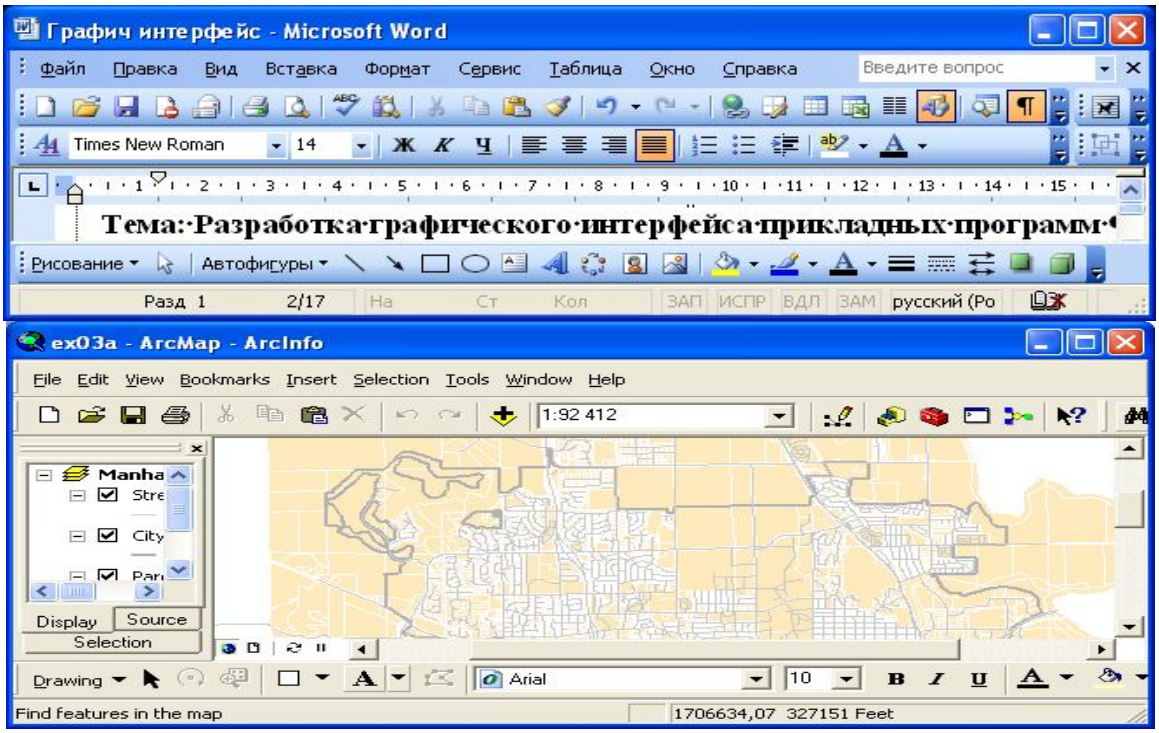


Рисунок 1.3 – Пример схожести вида графических интерфейсов MS Word и ArcMap

Проектирование интерфейса – процесс итерационный. На этом этапе разработки приложения желательно чаще общаться с пользователями и заказчиками приложения для выбора наиболее приемлемых по эффективности, удобству и внешнему виду интерфейсных решений.

Выбор того или другого типа интерфейса зависит от сложности приложения, поскольку каждый из них имеет свои недостатки, ограничения и предназначен для решения определенных заданий. При этом, необходимо отвечать на ряд вопросов: какое количество документов будет обрабатываться в проектируемом приложении и их тип, имеют ли данные древовидную структуру, будет ли нужна панель инструментов, какое количество документов обрабатывается за определенный интервал времени (например, за день) и тому подобное. Только после этого можно выбирать конкретный тип интерфейса.

При разработке интерфейса необходимо руководствоваться следующими принципами:

1. **Стандартизация.** Следует использовать стандартные, проверенные многими программистами и пользователями интерфейсные решения. Для Visual Basic это решение Microsoft. Под решениями имеются в виду

дизайн формы, расположение элементов управления на форме, их взаимное расположение, значки на кнопках управления, названия команд меню.

2. **Удобство и простота работы.** Интерфейс должен быть интуитивно понятным. Желательно, чтобы все действия легко запоминались и не требовали длительных процедур: выполнения дополнительных команд, лишних нажатий на кнопки, вызова промежуточных диалоговых окон.

3. **Внешний дизайн.** Нельзя, чтобы интерфейс утомлял зрение. Он должен быть рассчитан на длительную работу пользователя с приложением в течение дня.

4. **Неперегруженность формы.** Формы должны быть оптимально загружены элементами управления. При необходимости можно использовать вкладки или дополнительные страницы форм.

5. **Группирования.** Элементы управления в форме необходимо группировать по содержанию, используя элементы группирования: рамки, фреймы.

6. **Разреженность объектов формы.** Элементы управления следует располагать на определенном расстоянии, а не вплотную друг к другу. Для выделения элементов управления можно организовать пустые участки в форме.

В настоящее время для приложений, которые разрабатываются в среде Windows с помощью Visual Basic, используются три типа интерфейса:

1. Однодокументный SDI (Single-Document Interface).
2. Многодокументный MDI (Multiple-Document Interface).
3. Интерфейс типа «проводник» (Explorer).

Однодокументный интерфейс

Однодокументный интерфейс – это тип интерфейса, в котором предоставляется возможность работы только с одним документом в одном окне. Примером может служить редактор Microsoft WordPad. Интерфейс включает такие элементы:

- главное меню;
- панели инструментов с элементами управления;
- окно приложения для размещения элементов управления данными;
- элементы управления для работы с данными;
- строка состояния.

Многодокументный интерфейс

Главная особенность MDI интерфейса – можно многоразово открывать форму одного вида документа для нескольких разных по содержанию документов. Примером может служить редактор Microsoft Word. Интерфейс включает такие элементы:

- главное меню;
- панели инструментов с элементами управления;
- главное окно приложения (MDI-окно);
- дочерние окна приложения;
- элементы управления для работы с данными, расположенными в дочерних окнах;
- строка состояния.

Интерфейс типа «Проводник»

Интерфейс типа **Проводник** разрабатывается для доступа к иерархическим древовидным структурам, то есть к таким, где встречается вложенность. Примером вложенности могут служить папки и файлы. Файлы лежат в папках, которые, в свою очередь, лежат в вышестоящих папках и так далее. Примером такого интерфейса является проводник Windows. В этом файловом менеджере наглядно видна структура хранения папок и файлов, образующая иерархическое дерево. По своей сути это аналог интерфейса SDI, разработанный специально для древовидных структур.

Интерфейс приложения типа проводник содержит следующие элементы:

- главное меню;
- окно приложения для размещения элементов управления данными;
- иерархический список элементов древовидной структуры. Это могут быть папки и файлы, документы, если они организованы в иерархическую структуру;
- элементы управления для работы с данными: кнопки, поля, флажки;
- строка состояния.

Средства проектирования интерфейса

Трудно переоценить роль интерфейса пользователя в современном программном обеспечении. Удачный или неудачный интерфейс во многом предопределяет успех или неудачу всей разработки. Удобство интерфейса –

понятие достаточно субъективное. То, что нравится одному пользователю, совсем не подойдет для другого. То, что обеспечивает комфортную работу служащих одного отдела, будет встречено недоброжелательно в другом. Поэтому при проектировании системы лучше всего заранее включать в нее возможность настройки интерфейса под нужды различных категорий пользователей.

Создание приложений практически невозможно без использования элементов управления, так как они позволяют пользователю взаимодействовать с этими приложениями. С помощью элементов управления (меню **Вид / Панели инструментов / Элементы управления**) можно автоматизировать работу с документами. Представленные элементы управления позволяют разработать практически любые интерфейсы для пользователей.

Панель элементов управления – основной рабочий инструмент при визуальной разработке форм приложения (рис. 1.4).

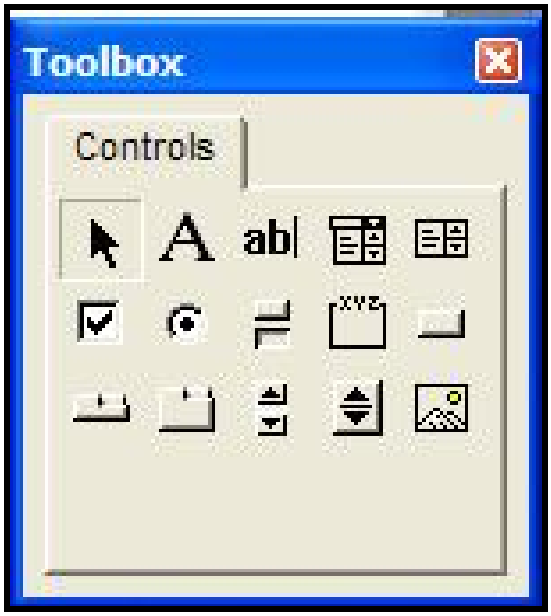


Рисунок 1.4 – Панель элементов управления в VBA

В составе панели элементов управления содержатся основные элементы управления форм – надписи, текстовые поля, кнопки, списки и другие элементы интерфейса (объекты), для быстрого визуального проектирования макета формы. В таблице 1.1 представлены названия и внешний вид объектов (элементов управления), которые встраиваются в форму.

Таблица 1.1 – Названия и внешний вид элементов управления

Тип объекта	Назначение	Графический вид
Label	надпись	
TextBox	текстове поле	
CommandButton	кнопка	
ChekBox	флажок	
OptionButton	переключатель	
Frame	группа (рамка)	
ListBox	список	
ComboBox	поле со списком	
Image	рисунок	
ShipButton	счетчик	
ScrollBar	полоса прокрутки	

Размещение элементов управления на форме осуществляется следующим образом:

- на панели **Toolbox** нажать кнопку необходимого элемента управления. Например, для размещения на форме текстовой надписи необходимо добавить кнопку **Label;**
- щелчком-протяжкой левой кнопкой мыши выбранный элемент переместить с панели на форму вместо возможного размещения объекта;
- для некоторых элементов, для придания нужного размера, необходимо не отпуская кнопку мыши, нарисовать рамку необходимого размера на форме и отпустить кнопку мыши.

В случае необходимости заданные при размещении элемента управления на форме размеры можно изменить позже, используя мышь или окно свойств объекта.

Элементы управления

Кнопка – элемент, при нажатии на который выполняется та или иная команда.

Флажок – элемент, который независимо от других может находиться в трех состояниях: включенном, выключенном (состояния могут быть определены как пользователем, так и программистом) и неактивном (определяется программой).

Поле (ввода текста) – дает пользователю возможность ввести текстовую информацию с целью последующей обработки в программе.

Переключатель – элемент, который может находиться во включенном, выключенном и неактивном состояниях. В отличие от флажка, если один из переключателей в группе включен, остальные включены быть не могут.

Список – средство выбора одного варианта из нескольких представленных. Допускается прокручивание списка, если не все его элементы видны одновременно.

Поле со списком – вставляет объект, являющийся сочетанием списка и поля. Пользователь может либо выбрать нужное значение из списка, либо ввести его в поле самостоятельно.

Счетчик – может передавать в программу свое значение от 1 до 100.

Полоса прокрутки – может передавать в программу число, равное расстоянию в пунктах от ее начала. Верхний предел расстояния неограничен.

Надпись (поле отображения текста) – отображает неизменяемый текст, например подпись к рисунку.

Рисунок – в этот элемент можно вставить рисунок из файла.

Некоторые общие свойства элементов управления

Ниже представлены общие свойства, которыми обладает большинство элементов управления (ЭУ).

Позиция. Позицию ЭУ определяют четыре свойства **Left**, **Top**, **Height**, **Width**. Свойства **Left**, **Top** задают координаты верхнего левого угла элемента управления. Свойства **Height** и **Width** – его высоту и ширину. Отсчет в системе координат ведется сверху вниз по оси **Y** и слева направо по оси **X.**

Цвет. Управление цветом осуществляется с помощью свойств **BackColor, FillColor, ForeColor,** которым по умолчанию назначаются стандартные цвета. Свойство **BackColor** отвечает за цвет фона и выбирается в диалоговом окне настройки цвета. Свойство **ForeColor** определяет цвет, используемый для отображения текста и графики в элементе управления, а свойство **FillColor** устанавливает цвет заполнения рисованных объектов.

Параметры шрифта. Вид шрифта в элементах управления выбирается при помощи установки значений свойства **Font.**

Доступность и видимость ЭУ. Для этого используются два свойства – **Enabled** и **Visible.** Свойство **Enabled** определяет, будет ли элемент управления реагировать на события (**True**) или нет (**False**). Свойство **Visible** позволяет сделать ЭУ невидимым (**False**).

Свойство **Name** является идентификатором элемента управления. Оно играет особую роль. Ошибка при его задании приводит к серьезным последствиям (невыполнению кода программы). Поэтому сначала всегда задается имя ЭУ, и лишь затем пишется для него процедура обработки события.

Вопросы для самоконтроля

1. В чем заключается основное достоинство Windows-приложений?
2. Что такое интерфейс и почему он является важнейшей частью любого приложения?
3. Опишите процесс проектирования интерфейса.
4. Какими принципами необходимо руководствоваться при разработке интерфейса?
5. Какие типы интерфейсов приложений под ОС Windows разрабатываются с помощью Visual Basic? Что вы можете сказать о каждом из них?
6. Какие средства в языке VBA используются для проектирания интерфейсов?
7. Перечислите и охарактеризуйте, какие элементы управления могут быть помещены на форму.
8. Какие свойства присущи элементам управления и какова технология их изменения?

1.4 ПЕРЕМЕННЫЕ И ТИПЫ ДАННЫХ

Операция – конструкция в языках программирования, аналогична по записи математическим операциям, то есть специальный способ записи некоторых действий. Наиболее часто применяются арифметические, логические и строковые операции. В отличие от функций, операции часто являются базовыми элементами языка и обозначаются различными символами пунктуации, а не алфавитно-цифровыми, они имеют специальный синтаксис и нестандартные правила передачи аргументов.

Фактически операция – это та же функция, но записываемая особым образом. По этой причине логично иметь возможность определять операции для произвольных типов данных таким же образом, как и методы – чтобы можно было работать с ними точно так же, как и с элементарными типами. Эта возможность называется «перегрузка операций» и присутствует в большинстве языков 4–5 поколений. В таких языках транслятор фактически подставляет вместо выполнения операции вызов соответствующей ей функции.

Операции делятся по количеству принимаемых аргументов на:

- унарные – один аргумент (отрицание, унарный минус)
- бинарные – два аргумента (сложение, вычитание, умножение)
- тернарные – три аргумента («условие ? выражение1 : выражение2»).

Арифметические операции задаются следующими символами (табл. 1.2).

Таблица 1.2 – Перечень арифметических операций

Знак	**Операция**
+	сложение
–	вычитание
*	умножение
/	деление
\	деление без остатка
mod	деление по модулю

Для записи операторов в языке VBA используются ключевые слова (зарезервированные слова), такие как **Dim**, **As, New, If**, **Then, Else, While, End.** При явном объявлении переменные определяются с помощью оператора **Dim.**

Dim Name As String – явное объявление типа данных.

Типы данных, используемых в Visual Basic, приведены в таблице 1.3.

Таблица 1.3 – Типы данных

Тип данных	**Размер в байтах**	**Описание типа**	**Диапазон**
1	**2**	**3**	**4**
Byte	1	Однобайтное целое число	от 0 до 255
Integer	2	Короткое целое число	от -32768 до 32767
Long	4	Длинное целое	± 21474833648

Продолжение таблицы 1.3

1	2	3	4
Double	8	Двойное с плавающей точкой	от $5{,}0\times10^{-324}$ до $1{,}7\times10^{308}$
Single	4	Десятичные числа одинарной точности	от $1{,}5\times10^{-45}$ до $3{,}4\times10^{38}$
String	1 байт на каждый символ	Строка переменной длины	от 0 до 2×10^{9}
Boolean	2	Логическое значение **True** или **False**	
Currency	8	Число в денежном формате	
Date	8	Дата от 1 января 100 г. до 31 декабря 9999 г.	
Variant	>=16 байтов	Любой тип данных	

Объявление и использование переменных

В программе, прежде чем использовать переменную, ее надо объявить. В языке Visual Basic для описания переменных используются специальные операторы. Переменными обозначаются значения, изменяемые в ходе выполнения программы. В языке VBA каждая переменная характеризуется своим типом (**type**). От типа зависит, какие значения принимает переменная. Объявление переменной определяет ее тип.

В простейшем случае синтаксис объявления переменной состоит из одного выражения, включающего в себя:

- ключевое слово **Dim;**
- имя переменной;
- ключевое слово **As;**
- название типа переменной.

Объявление переменной в кодах программы выглядит следующим образом.

Dim *ИмяПеременной* **As** тип

В одной строке можно объявить сразу несколько переменных.

Dim *ИмяПеременной1* **As** тип1, *ИмяПеременной2* **As** тип2,...

Имя переменной должно быть уникальным в пределах программного кода.

ПРИМЕР.

Dim *intЧисло* **As Integer,** *Строка* **As String**

где переменная *intЧисло* – переменная целого типа, может принимать значения от -32768 до 32767;

переменная *Строка* – переменная строкового типа.

Dim *strA* (1 to 33) **As String** – объявление строкового массива, содержащего 33 элемента.

В случае если тип переменной не объявлен, то переменная считается переменной универсального типа **Variant** и для ее хранения отводится в памяти максимальное количество байт (16), что приводит к неэффективному использованию памяти и замедлению работы программы.

Недостатком языка VBA является то, что он позволяет программистам обходиться без объявления переменных. Такой порядок используется в языке VBA по умолчанию. По мере необходимости в программу добавляются новые переменные без их объявления.

Однако добавление переменных без их объявления приводит к возникновению ряда ошибок. Например, если случайно неправильно набрано имя переменной, программа просто воспримет это как добавление новой переменной с другим именем. В процессе выполнения программы могут возникнуть проблемы. Кроме того, если переменная не объявлена, для нее автоматически будет выбран тип **Variant.** В результате доступные системные ресурсы будут использованы крайне нерационально.

При объявлении переменных VBA автоматически распознает неправильно набранные имена переменных, что облегчает устранение ошибок. Кроме того, для объявленных переменных можно подобрать такие типы данных, которые наилучшим образом будут подходить для сохранения обрабатываемых значений.

Инструкция Option Explicit

Чтобы активировать объявление переменных, требуется добавить выражение **Option Explicit** в каждом модуле перед началом всех процедур. Для активизации необходимости объявления переменных можно воспользоваться меню **Tools / Options** (Сервис / Параметры). В открывшемся диалоговом окне на вкладке **Editor** (Редактор) необходимо установить флажок опции **Require Variable Declaration** (Объявление переменных обязательно).

При использовании инструкции **Option Explicit** необходимо явно описать все переменные с помощью инструкций **Dim, Private, Public,**

ReDim или **Static.** При попытке использовать неописанное имя переменной возникает ошибка во время компиляции.

Для автоматической вставки в программу инструкции **Option Explicit** откройте редактор VBA (быстрые клавиши Alt+F11) войдите в меню **Сервис / Параметры** в открывшемся диалоговом окне **Параметры** установите флажок **Явное описание переменных.**

Объявление констант

Переменные, значения которых не меняются в процессе выполнения программы, называются константами.

Const *ИмяКонстанты* [**As Тип**] = ЗначениеКонстанты

Константы (**constant**) – это данные, которые не изменяются в ходе выполнения программы. В языке VBA различают константы двух типов: литеральные (**literal**) и символьные (**symbolic**). Литеральные константы набираются непосредственно в кодах программы и могут представлять собой числа и строки (текстовые значения). Строковые константы должны заключаться в двойные кавычки.

ПРИМЕР.

```
МояСтрока = "четыре"
МоеЧисло = 4
```

Что касается числовых значений, VBA распознает их экспоненциальную запись и (с использованием префикса &H) шестнадцатеричную запись.

```
1.2E5 ' число 1,2, умноженное на 10 в пятой степени, т.е. 120000
&HFE ' эквивалентно десятичному числу 254
```

Шестнадцатеричная система исчисления основана на использовании 16 цифр, тогда как привычная нам десятеричная система построена на применении десяти цифр. В шестнадцатеричной системе цифры от 0 до 9 обозначают первые десять цифр (как и в десятичной системе), а для обозначения последующих шести цифр используются первые шесть букв латинского алфавита (от A до F).

Символьные константы имеют собственные имена. Если в коде программы возникает необходимость в использовании значения такой константы, следует указать ее имя. Определяются символьные константы с

помощью ключевого слова **Const.**

Const *ИмяКонстанты = ЗначениеКонстанты*

В качестве имени константы можно использовать любое имя, допустимое в языке VBA. Значение константы – это литеральная, числовая либо строковая константа, определяющая значение создаваемой символьной константы.

ПРИМЕР.

Const МОЕИМЯ = «*Петр*»
Const МОЙВОЗРАСТ = 30

Символьные константы обладают двумя преимуществами. Имя такой константы может и должно быть описательным, что значительно упрощает чтение кодов программы. Однако более важно следующее: если возникнет необходимость изменить значение константы в кодах программы, то достаточно будет сделать это только в том месте, где она была объявлена. Согласно общепринятому соглашению, имена констант набираются исключительно прописными буквами, тогда как имена других элементов, например переменных, состоят из комбинаций прописных и строчных букв. Например, имя **ПРОЦЕНТНАЯСТАВКА** соответствует константе, а имя **ПроцентнаяСтавка** – переменной. Для программы VBA это правило не имеет никакого значения, однако, следуя ему, можно довольно просто отличить в кодах программы имена констант от имен переменных.

ПРИМЕР.

Const *Pi* **As Single** = 3,14

Если константа может использоваться во всех модулях и процедурах, то сначала нужно написать зарезервированное слово **Public**.

Public Const *MyConst* **As Integer** = 59

Если константа может использоваться только в данном модуле, то сначала нужно записать слово Private.

Private Const *Stavka* **As Single** = 0,007

Присваивание

Переменная может получить или изменить значение с помощью оператора присваивания:

[Let] *ИмяПеременной* = **Выражение**

ПРИМЕР. Определить, какие значения получат переменные после выполнения следующей программы.

```
Dim Example As Single, a As Integer, b As Byte, c As Integer
Private Sub Command1_Click()
Example = 5,8
a = 5
b = 7
c = a+b
a = 100
End Sub
```

После выполнения программы значения переменных будут следующие a = 100, b = 7, c = 12, Example = 5,8.

Преобразование типов данных

Преобразование в VBA содержит процедуры, которые преобразовывают значения одного типа в значения другого типа. Так, функция **Val** преобразует строки в число, а **Str** преобразует числа в строку. **Hex, Oct** преобразуют числа в другую систему счисления. **CBool**, **CByte**, **CCur**, **CDbl, CDec, CInt, CLng, CSng, CStr, CVar, Fix**, **Int** – один тип данных в другой (табл. 1.4). Например, **CDbl** – в тип **Double.**

Таблица 1.4 – Функции преобразования типов данных

Функция	**Тип результата**	**Префикс типа**	**Описание типа**
CBool (x)	Boolean	bln	Логическое значение
CByte (x)	Byte	byt	Однобайтное целое число (от 0 до 255)
CInt (x)	Integer	int	Короткое целое число (до ±32 тыс.)
CCur (x)	Currency	cur	Число с фиксированной точкой (денежный тип)
CDate (x)	Date	dtm	Дата и время
CDbl (x)	Double	dbl	Число с плавающей точкой двойной точности (Целые, дробные 10±348)
CLng (x)	Long	lng	Длинное целое (± 2 млрд.)
CSng (x)	Single	sng	Число с плавающей точкой одинарной точности
CStr (x)	String	str	Текстовая строка
CVar (x)	Variant	var	Любое значение из перечисленных выше

Организация диалога с пользователем

Функция **InputBox** используется для получения информации от пользователя. Синтаксис этой функции следующий:

InputBox (*Message, Title, Default, y, x*)

где *Message* – сообщение-подсказка;

Title – заголовок;

Default – значение по умолчанию;

y, *x* – координаты расположения окна.

Функция **MsgBox** используется для выдачи информации пользователю. Синтаксис этой функции следующий:

MsgBox (*Msg, Style, Title, Help, Ctxt*)

где *Msg* – сообщение для пользователя;

Style – вид кнопки (например, vbOKOnly – отображается только кнопка «OK»);

Title – заголовок;

Help – имя файла справки (необязательный аргумент). Если этот аргумент указан, необходимо указать номер раздела справки.

Вопросы для самоконтроля

1. Охарактеризуйте такую конструкцию в языках программирования, как операция.

2. Какие ключевые слова используются в языке VBA для записи операторов?

3. Каким образом в VBA объявляется тип переменной и какие типы данных вы знаете?

4. Что такое константа и каким образом она объявляется в VBA?

5. Каких типов константы вы знаете?

6. В каких целях используется оператор присваивания? Приведите примеры.

7. С помощью каких функций в VBA происходит преобразование типов данных?

8. В каких целях в VBA используется инструкция **Option Explicit**?

9. С помощью каких функций осуществляется диалог с пользователем в VBA?

1.5 БАЗОВЫЕ КОНСТРУКЦИИ ЯЗЫКА VBA

В языке VBA содержится мощный набор инструментов, предназначенный для управления данными (это важно, поскольку каждая программа обрабатывает данные того или иного типа).

Понятие **синтаксиса** в языках программирования имеет много общего с аналогичным понятием обычных языков (русского, английского). Так называется совокупность правил, определяющая порядок использования и организации элементов данного языка. Поэтому прежде, чем приступить к написанию собственного кода VBA, необходимо ознакомиться с синтаксисом языка программирования.

Коды программы VBA состоят из выражений (**statements**). В каждой строке кода, как правило, располагается одно отдельное выражение. Но данное правило имеет исключения:

- несколько выражений можно писать в одной строке и разделить их двоеточием;
- можно использовать символ продолжения строки (пробел, за которым следует символ подчеркивания). Это позволяет создавать выражение, занимающее несколько строк.

Не рекомендуется использовать первый прием для расположения нескольких выражений в одной строке, поскольку реальной пользы это не принесет, но сами коды становятся более сложными для чтения и восприятия.

Использование символа продолжения строки позволяет записывать длинные строки кода на нескольких строках экрана. Чтобы разбить строку в определенной позиции, необходимо нажать клавишу **Пробел,** затем символ подчеркивания (_) а затем нажать клавишу **Enter.**

Не допускается использовать символ продолжения строки только внутри заключенных в кавычки строк. Рекомендуется продолжение каждой строки выделять отступом слева. Для VBA это не имеет никакого значения, но зато при чтении кодов программы будет понятно, что это продолжение предыдущей строки, а не начало новой.

Комментарии в программном коде

В кодах VBA комментариями (**comments**) называется текст, который не влияет на ход выполнения программы. Комментарии используются для добавления пояснений в программу, например, значения каких аргументов

требуется передавать процедурам. Любые комментарии-пояснения могут оказаться полезными в будущем разработчику программы или другим программистам, изучающим или исправляющим этот код.

Один из способов создания комментариев состоит в использовании символа апострофа (единичные кавычки). Все, что набрано после апострофа и до конца текущей строки, считается комментарием. Символ апострофа может быть набран как в начале строки, так и внутри нее.

Dim MyWB **As Single** *'Это комментарий*

Для обозначения комментариев можно также использовать ключевое слово **REM** (образовано от слова **remark** – замечание). Слово **REM** обязательно должно быть указано в начале строки:

REM *Это комментарий*

Dim MyWB **As Single REM** *А это не комментарий*

В редакторе VBA комментарии отображаются другим цветом – по умолчанию зеленым.

Символ апострофа можно использовать с целью перевода в разряд комментариев отдельных фрагментов кода программы. Такая возможность оказывается полезной в тех случаях, когда требуется проверить, как будет работать программа без отдельных, уже набранных строк кода. Не удаляя эти строки, а просто превращая их в комментарии. Далее, если необходимо вновь вернуть строку в программный код, нужно удалить набранный перед ней символ апострофа.

Форматирование кодов

Символы пробелов (обычные пробелы, символы табуляции и пустые строки программы) игнорируются при выполнении программы VBA. Их можно использовать для визуального форматирования кодов, чтобы сделать их более удобными для чтения и восприятия. Пустые строки визуально отделяют фрагменты кодов, функционально не зависящие друг от друга. Наглядность программы достигается также путем смещения выражений вправо относительно других выражений с помощью табуляции.

Правила выбора имен объектов в языке VBA

Эти правила в языке VBA распространяются на символьные кон-

станты, переменные, свойства и практически на любые другие элементы, которым в кодах программы могут присваиваться имена. В языке VBA имена должны соответствовать следующим требованиям:

- начинаться с букв;
- состоять не более чем из 255 символов;
- не совпадать с ключевыми словами VBA;
- не содержать в себе точек, пробелов, а также символов !, @, #, &, –, $.

Необходимо давать переменным и прочим элементам описательные имена, по которым можно было бы определить их назначение в контексте программы.

Имена, состоящие из нескольких слов, можно образовать путем чередования строчных и прописных букв.

ПРИМЕР. *Процентная_ставка* либо *ПроцентнаяСтавка*

Поскольку использование пробелов не допускается, то вместо пробелов можно использовать символ подчеркивания. Регистр символов не имеет значения. Поэтому имена **Сумма**, **сумма** и **СУММА** считаются в языке VBA идентичными.

Конструкции языка VBA

Как и во всех других языках программирования, в VBA имеются различные управляющие конструкции, позволяющие изменять порядок выполнения программы. Без использования управляющих конструкций будет происходить последовательное выполнение операторов языка программирования от первого до последнего (линейный процесс). Хотя в некоторых самых простых случаях и этого бывает достаточно, однако обычно все-таки требуется изменять порядок выполнения операторов, либо пропуская выполнение некоторых из них, либо, наоборот, многократно повторяя. Оказывается, для реализации любых алгоритмов достаточно иметь только два вида инструкций управления: ветвления и циклы.

Ветвление

Управляющие конструкции ветвления позволяют проверить некоторое условие, а затем в зависимости от результатов этой проверки выполнить ту или иную группу операторов. Для организации ветвлений в VBA используются различные формы оператора ветвления **If** и оператор выбора **Select Case**.

Простейшая, краткая форма оператора **If** используется сначала для

проверки одного условия, а затем в зависимости от результата этой проверки либо для выполнения, либо для пропуска одного оператора или блока из нескольких операторов. Краткая форма оператора ветвления **If** может иметь как однострочную, так и блочную форму. Запись в одну строку краткой формы **If** имеет следующий вид:

If Условие **Then** Оператор1 **Else** Оператор2

Полная форма оператора **If** используется в тех случаях, когда имеются два различных блока операторов и по результатам проверки условия нужно выполнить один из них. Такая форма **If** не может записываться в одну строку и всегда имеет блочную форму записи. В блочной форме краткое ветвление выглядит следующим образом:

If Условие **Then**
    Оператор
End If

В качестве условия можно использовать логическое выражение, возвращающее значения **True** (Истина) или **False** (Ложь), или любое арифметическое выражение. Если используется арифметическое выражение, то нулевое значение этого выражения эквивалентно логическому значению **False,** а любое ненулевое выражение – значению **True**. В том случае, когда условие возвращает значение **False**, оператор или блок операторов, заключенных между ключевыми словами **Then** и **End If** и составляющих тело краткого оператора ветвления, не будет выполняться.

If Условие **Then**
    Оператор1
Else
    Оператор2
End If

Если условие истинно, выполняется первый блок операторов, заключенный между ключевыми словами **Then** и **Else;** в противном случае выполняется второй блок, заключенный между ключевыми словами **Else** и **End If.**

Иногда приходится делать выбор одного действия из целой группы альтернативных действий на основе проверки нескольких различных

условий. Для этого можно использовать цепочку операторов ветвления **If**...**Then**...**ElseIf.**

```
If Условие1 Then
    Операторы1
  ElseIf  Условие2 Then
    Операторы2
…
  ElseIf  УсловиеN Then
    ОператорыN
End If
```

Цепочки операторов **If**...**Then**...**ElseIf** обладают большой гибкостью и позволяют решить все проблемы, однако если выбор одной из нескольких возможностей все время основывается на различных значениях одного и того же выражения, гораздо удобнее использовать специально предназначенный для этого оператор выбора **Select Case,** имеющий следующий синтаксис:

```
Select  Case  Условие Case  СписокЗначений1
        Операторы1> Case  СписокЗначений2
        Операторы2 Case  СписокЗначений3
        Операторы3
...
   Case  Else
        Операторы_Else
End  Select
```

Проверяемое выражение вычисляется в начале работы оператора **Select Case** и может возвращать значение любого типа, например логическое, числовое или строковое.

Список выражений содержит одно или несколько выражений, разделенных запятой. При выполнении оператора проверяется, соответствует ли хотя бы один из элементов этого списка проверяемому выражению. Элементы списка условий могут иметь одну из следующих форм.

Условие

В этом случае проверяется, совпадает ли значение проверяемого условия с этим условием.

```
Условие1 To  Условие2
```

В этом случае проверяется, находится ли значение проверяемого условия в указанном диапазоне значений.

Is ЛогическийОператор Условие

В этом случае проверяемое условие сравнивается с указанным значением с помощью заданного логического оператора (например, условие **Is** > = 10 считается выполненным, если проверяемое значение не меньше 10).

Если хотя бы один из элементов списка соответствует проверяемому условию, то выполняется соответствующая группа операторов и на этом выполнение оператора **Select Case** заканчивается, а остальные списки выражений не проверяются, то есть в этом случае отыскивается только первый подходящий элемент списков выражений. Если ни один из элементов всех этих списков не соответствует значению проверяемого выражения, то выполняются операторы группы **Else** (если такая присутствует).

Циклы

В VBA имеется богатый выбор средств организации циклов, которые можно разделить на две основные группы: циклы с условием **Do**...**Loop** и циклы с перечислением **For**...**Next**. Циклы типа **Do**... **Loop** используются в тех случаях, когда заранее не известно, сколько раз должно повториться выполнение блока операторов, составляющего тело цикла. Такой цикл продолжает свою работу до тех пор, пока не будет выполнено определенное условие. Существуют четыре вида циклов **Do**...**Loop**, которые различаются типом проверяемого условия и временем выполнения этой проверки. Синтаксисы этих четырех конструкций приведены в таблице 1.5.

Таблица 1.5 – Синтаксисы управляющих конструкций

Синтаксис конструкции	**Описание**
1	**2**
Do While Условие Операторы тела цикла **Loop**	Условие проверяется до выполнения группы операторов, образующих тело цикла. Цикл продолжает свою работу, пока это условие выполняется (имеет значение **True**), иными словами, в этой конструкции указывается условие продолжения работы цикла

Продолжение таблицы 1.5

1	2
Do Операторы тела цикла **Loop While** Условие	Условие проверяется после выполнения хотя бы одиного раза операторов, составляющих тело цикла. Цикл продолжает свою работу, пока это условие остается истинным, иными словами, в этой конструкции указывается условие продолжения работы цикла
Do Until Условие Операторы тела цикла **Loop**	Условие проверяется до выполнения группы операторов, образующих тело цикла. Цикл продолжает свою работу, если это условие еще не выполнено, и прекращает работу, когда оно станет истинным, иными словами, в этой конструкции указывается условие прекращения работы цикла
Do Операторы тела цикла **Loop Until** Условие	Условие проверяется после выполнения хотя бы один раз операторов, составляющих тело цикла. Цикл продолжает свою работу, если это условие еще не выполнено, а когда оно станет истинным, цикл прекращает работу, иными словами, в этой конструкции указывается условие прекращения работы цикла

В VBA также есть два вида операторов цикла с перечислением **For**...**Next**. Очень часто при обработке массивов, а также в тех случаях, когда требуется повторить выполнение некоторой группы операторов заданное число раз, используется цикл **For**...**Next** со счетчиком. В отличие от циклов **Do**... **Loop** данный цикл использует специальную переменную, называемую счетчиком, значение которой увеличивается или уменьшается при каждом выполнении тела цикла на заданную величину. Когда значение этой переменной достигает заданного значения, выполнение цикла заканчивается.

Синтаксис такого цикла имеет следующий вид (в квадратные скобки заключены необязательные элементы синтаксической конструкции):

For счетчик = начальное Значение **To** конечное Значение
  [**Step** приращение]
  Операторы тела цикла
Next счетчик

Рассмотрим еще один вид цикла **For**... **Next**, часто используемый в VBA при обработке объектов, составляющих массив или семейство однородных объектов. В цикле **For Each**... **Next** счетчик отсутствует, а тело

цикла выполняется для каждого элемента массива или семейства объектов. Синтаксис такого цикла имеет следующий вид. В квадратных скобках указаны необязательные операторы.

For Each элемент **In** совокупность
Операторы тела цикла **Next** [элемент]

Здесь **элемент** – это переменная, используемая для ссылки на элементы семейства объектов; **совокупность** – имя массива или семейства.

ПРИМЕР. Выдача на печать списка всех полей для всех таблиц текущей открытой базы данных. Программный код решения данной задачи приведен ниже.

```
Public Sub EnumerateAllFields( )
Dim MyBase As Database
Dim t As TableDef, f As Field
    Set MyBase=DBEngine.Workspaces(0).Databases(0) For Each t In MyBase.TableDefs
        Debug.Print "Таблица: " & t.Name For Each f In t.Fields
        Debug.Print "Поле: " & f.Name Next f Next t
    Set MyBase = Nothing
End Sub
```

В операторах **Dim** переменная **MyBase** объявлена как объект «База данных DAO», а переменные **t** и **f** – как определение таблицы и поле таблицы соответственно. Оператор **Set** назначает переменной **MyBase** текущую открытую базу данных. Далее для каждого определения выполняется вывод на печать названия таблицы, а затем вложенный цикл такого же типа, печатает названия всех ее полей.

Массивы

Массивы позволяют сохранять в одном месте наборы элементов данных. Массив имеет имя. Доступ к его элементам осуществляется с помощью числовых индексов. Элементы массива имеют одинаковый тип данных VBA. Предположим, что программа должна обработать итоговые значения объемов продаж по каждому из 12 месяцев прошедшего года. Можно создать массив из 12 элементов и значение данного показателя за январь сохранить, например, как первый элемент этого массива, значение по февралю – как второй элемент и т.д. Использование массивов для работы с данными во многих случаях упрощает задачу написания кодов программы

и делает их более удобными для чтения и восприятия. В VBA предусмотрено использование массивов двух типов – статических и динамических.

Статические массивы

Количество элементов статического массива строго определено. Размер массива (количество его элементов) определяется при его объявлении. Указанный размер остается неизменным в течение всего процесса выполнения программы.

Синтаксис объявления массива имеет следующий вид:

Dim ИмяМассива(n) **As** Тип

Имя массива должно удовлетворять стандартным правилам присвоения имен языка VBA, а **тип** может быть любым предусмотренным в языке VBA типом данных. Размер массива определяется параметром **n.**

В действительности размер массива будет на единицу большим числа **n,** поскольку нумерация элементов массивов VBA начинается с индекса 0. Поэтому массив, объявленный как

Dim МойМассив(50) **As** Integer

состоит из 51 элемента. Все они будут пронумерованы индексами от 0 до 50. Для доступа к элементам массива необходимо указать имя массива и заключенное в круглые скобки значение индекса.

```
МойМасив (0) = 1
МойМассив (25) = 73
```

В качестве индекса указываются целочисленные переменные или константы.

```
Dim idx As Integer
idx = 12
МойМассив (idx) = 44  'Равнозначно использованию записи
МойМассив (12) = 44
```

Чтобы отсчет индексов массива начинался не с нуля, можно воспользоваться одним из двух вариантов.

Вариант 1. В начале модуля за пределами всех процедур надо добавить выражение **Option Base 1.** Во всех массивах данного модуля отсчет

индексов будет начинаться с единицы, а не с нуля.

Вариант 2. В объявлении массива использовать ключевое слово **To**.

Dim ИмяМассива (*НижняяГраница* **To** *ВерхняяГраница*) **As** Тип

Значения *НижняяГраница* и *ВерхняяГраница* должны быть целыми, причем первое значение обязательно должно быть меньше второго. Например.

Dim МойМассив (20 **To** 40) **As** Long

Массивы с одним индексом называются одномерными. Чтобы создать массив с двумя или нескольким индексами, необходимо указати их в момент объявления массива.

Dim Двухмерный (10, 10) **As** Single

Такой массив будет состоять из 121 элемента, он начинается элементом **Двухмерный (0, 0)** и заканчивается элементом **Двухмерный (10, 10)**. Первая размерность включает в себя 11 элементов, вторая – также 11. Общее количество элементов массива вычисляется как произведение числа 11 на число 11. При объявлении многомерных массивов можно использовать ключевое слово **To.**

Dim ШахматнаяДоска (1 **To** 8, 1 **To** 8) **As** String

В VBA не предусмотрено никаких ограничений на максимально допустимое количество элементов массива и на его размерность. На практике, однако, и первая, и вторая характеристика ограничены доступными объемами системной памяти и дискового пространства, но эти ограничения вряд ли когда-нибудь смогут помешать при выполнении програмного кода. В случае если компьютер работает довольно медленно по причине недостаточного количества доступной памяти или дискового пространства, работа с массивами больших размеров может значительно увеличить время выполнения программы.

Если одна из процедур попытается обратиться к несуществующему элементу массива, это вызовет ошибку выполнения программы.

ПРИМЕР. Если массив объявлен как:

Dim МойМассив (20) **As** Integer

то при вводе выражения **МойМассив (27) = 10** либо **МойМассив (30) = 20** возникнет ошибка.

Динамические массивы

Размер динамического массива не является фиксированным. В ходе выполнения программы он может увеличиваться и уменьшаться по мере необходимости. Динамический массив объявляется с помощью пустых скобок.

Dim ДинамическийМассив () **As** Тип

Прежде чем приступить к использованию массива, необходимо определить его размер с помощью выражения **ReDim**.

ReDim Динамический массив (размер)

Аргумент размер определяет как размерность массива, так и количество элементов в каждой размерности.

ПРИМЕР. Динамический массив.

```
Dim Динамический1() As String
Dim Динамический2() As Integer
Dim Динамический3() As Object
…
ReDim Динамический1(100,)          ' 1 размерность, 101 элемент
ReDim Динамический2(-5 To 5)       ' 1 размерность, 11 элементов с -5 по 5
ReDim Динамический2(5, 5, 5)       ' 3 размерности, всего 216 элементов
```

Выражение **ReDim** не позволяет изменять тип динамического массива, однако с его помощью можно изменять размер этого массива любое количество раз. Как правило, при использовании выражения **ReDim** все прежние значения массива теряются. Чтобы сохранить уже имеющиеся данные (по возможности), необходимо добавить к выражению **ReDim** ключевое слово **Preserve**.

ReDim Preserve Динамический(100)

Существуют следующие ограничения на сохранение данных с помощью ключевого слова **Preserve:**

- если размер массива уменьшается, значения отбрасываемых элементов теряются;

- для многомерных массивов можно изменять только объем последней размерности;
- можно изменять количество размерностей.

Размер массива

В VBA существуют две функции, которые позволяют определить размеры любого массива. В частности, они предоставляют возможность определить наименьший и наибольший допустимый индекс массива.

UBound (ИмяМассива, размерность)

LBound (ИмяМассива, размерность)

Функция **UBound** возвращает наибольший, а функция **LBound** – наименьший допустимый индекс заданного массива. Аргумент **размерность** является необязательным, он определяет, индекс какой размерности массива требуется вернуть. По умолчанию его значение равно единице. Например:

```
Dim МойМассив(1 To 5, 10 To 25)
x = LBound(МойМассив)      ' Возвращает 1
x = UBound(МойМассив, 2) ' Возвращает 25
```

Вопросы для самоконтроля

1. Объясните понятие синтаксиса в языках программирования?
2. Каким образом осуществляется перенос строк кода с одной на другую?
3. Для чего используются комментарии в программном коде и каким образом они создаются?
4. Для чего при написании кода используется форматирование и обязательно ли его необходимо использовать?
5. Каковы основные правила выбора имен для элементов в языке VBA?
6. Для чего используются управляющие конструкций в языке VBA?
7. Для чего используется ветвление, каким образом оно организовано в VBA? Приведите примеры.
8. Какие формы операторов ветвления вы знаете. Приведите примеры.
9. Какие средства организации циклов в VBA вы знаете?
10. На какие группы можно разделить циклы в VBA? Приведите примеры.
11. Для каких целей используются массивы? Как они организуются в VBA?
12. Какие типы массивов существуют в VBA? Приведите примеры.

1.6 ПРИНЦИПЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Основу структурного программирования составляют такие положения:

- алгоритмы и программы должны состоять из небольшого набора базовых конструкций;
- алгоритмы целесообразно строить по шагам по принципу «сверху вниз», то есть от укрупненной схемы к детализированной;
- сложную задачу целесообразно разбивать на простые подзадачи, а программу составлять из модулей – независимых частей, которые имеют простую структуру и допускают независимую отладку.

Структурированными будем называть алгоритмы, которые построенны из базовых конструкций, а процесс их разработки – структурированием. Базовые алгоритмические конструкции имеют фиксированную структуру, один вход и один выход. Каждая конструкция предусматривает выполнение определенного действия и реализуется одним оператором алгоритмического языка. Обобщенные структуры базовых конструкций алгоритмов и программ приведенные на рисунке 1.5.

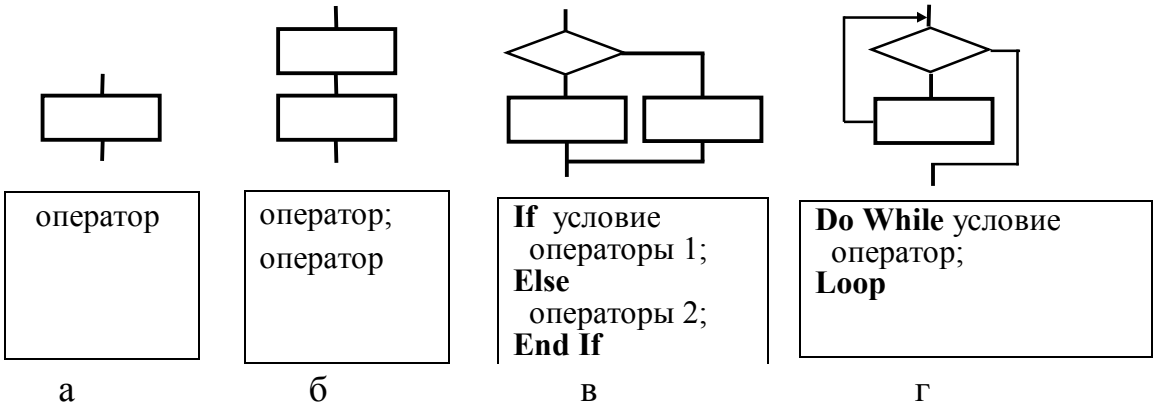
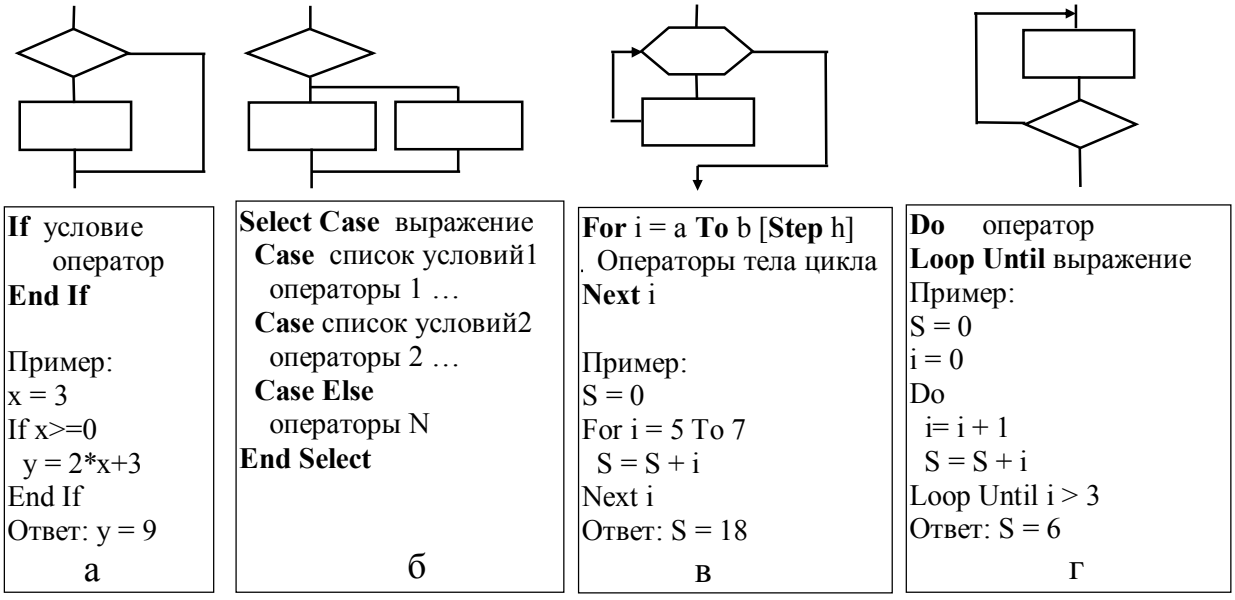


Рисунок 1.5 – Базовые конструкции алгоритмов:
а – функциональный блок; б – конструкции: следование и составной оператор; в – выбор и условный оператор; г – цикл с предусловием, оператор цикла While

Элементарной структурной единицей алгоритма является функциональный блок, а в программе – простой оператор (рис. 1.5, а). Этот блок предусматривает один шаг преобразования, обработки или отображения информации.

Доказано, что алгоритм решения любой задачи можно построить с использованием конструкций: **следование**, **выбор** и **цикл с предусловием** (рис. 1.5, б, в, г).

Для более полного использования возможностей алгоритмических языков набор базовых конструкций расширен (рис. 1.6). В этот набор дополнительно включены конструкции: **выбор с обходом** (а), **выбор варианта** (б), **цикл с параметром** (в) и **цикл с постусловием** (г). Рассмотренные конструкции составляют набор базовых конструкций, которые используются для конструирования алгоритмов и написания программ.



If условие
оператор
End If

Пример:
x = 3
If x>=0
y = 2*x+3
End If
Ответ: y = 9

а

Select Case выражение
Case список условий1
операторы 1 …
Case список условий2
операторы 2 …
Case Else
операторы N
End Select

б

For i = a **To** b [**Step** h]
Операторы тела цикла
Next i

Пример:
S = 0
For i = 5 To 7
S = S + i
Next i
Ответ: S = 18

в

Do оператор
Loop Until выражение
Пример:
S = 0
i = 0
Do
i= i + 1
S = S + i
Loop Until i > 3
Ответ: S = 6

г

Рисунок 1.6 – Расширенные конструкции алгоритмов:
а – выбор с обходом; б – выбор варианта; в – цикл с параметром;
г – цикл с постусловием

На рисунке 1.6 показаны сложные алгоритмические конструкции, которые реализуют простую последовательность действий (а), последовательность с простым разветвлением (б) и последовательность с простым циклом (в).

Конструкции со сложными разветвлениями (на 2 и больше ветвей) и сложными циклами (цикл в цикле) строятся путем замещения функциональных блоков другими конструкциями, например, после замещения функциональных блоков S3 (рис. 1.7, б) и S2 (рис. 1.7, в) конструкциями **выбор** и **цикл**, получим более сложные алгоритмические и программные конструкции (рис. 1.7).

В новых конструкциях предшествующие связи сохраняются. Аналогично можно получить и более сложные алгоритмические и программные конструкции. Таким образом, использование в алгоритмах базовых конструкций упрощает программирование, поскольку любая из них реализуется соответствующим оператором алгоритмического языка.

Метод пошаговой детализации

Разработку алгоритмов сложных задач целесообразно выполнять методом пошаговой детализации. В основу метода положен принцип декомпозиции: сложные действия делятся на последовательность простых действий, а сложные задачи – на простые подзадачи. Конструирование алгоритмов выполняется с использованием базовых конструкций и вспомогательных алгоритмов, которые предназначены для решения подзадач.

Алгоритм разрабатывается из одного или нескольких шагов в зависимости от сложности задачи, которая решается. При этом используются два способа конструирования алгоритмов:

1) пошаговое уточнение алгоритма;
2) использование вспомогательных алгоритмов.

Пошаговое уточнение алгоритма

Данный способ основывается на правилах построения сложных алгоритмических конструкций с использованием базовых конструкций (рис. 1.7).

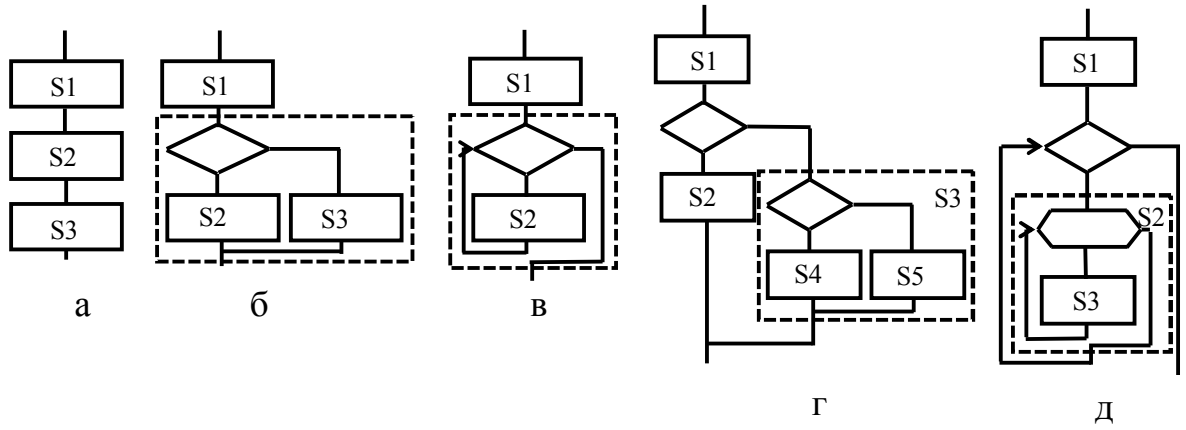


Рисунок 1.7 – Сложные алгоритмические конструкции
с использованием базовых конструкций

Первый шаг процесса решения задачи делится на крупные действия и разрабатывается укрупненная схема алгоритма. Каждое действие запи-

сывается в виде четкой инструкции, которая задает, **что делать** (но не как делать). Укрупненная схема показывает общий порядок решения задачи и имеет вид простой последовательности функциональных блоков и других конструкций (рис. 1.7, а, б, в).

На втором и последующих шагах выполняется детализация алгоритма. Сложные действия уточняются и детализируются. Для любого из них определяются свои последовательности действий и базовых конструкций, которые определяют, **как надо** выполнять сложные действия. Этими базовыми конструкциями заменяют функциональные блоки укрупненной схемы алгоритма (рис. 1.7, г, д).

Детализация заканчивается на том шаге, после которого алгоритм становится пригодным для программирования, то есть, когда каждая конструкция алгоритма может быть представлена в программе одним оператором алгоритмического языка. В результате такого подхода алгоритм будет складываться только из базовых конструкций.

Порядок конструирования алгоритмов путем пошагового уточнения рассмотрим на примере задачи табулирование функции.

ПРИМЕР. Используя пошаговую детализацию, решим задачу по определению дальности стрельбы, которая вычисляется по формуле:

$$D = \frac{V^2 \sin 2\alpha}{g},$$

где D – дальности стрельбы;

V – скорость полета;

α – угол стрельбы;

q – ускорение свободного падения

Необходимо разработать алгоритм вычисления значений функции $D = f(V,\alpha)$, если скорость $V = \{v1, v2,..., vn\}$, где $n = 5$, а угол стрельбы α изменяется от $\alpha 1$ к $\alpha 2$ с шагом $\Delta\alpha$.

Метод решения. Поскольку V – это массив, то вычисление выполняются по следующей формуле:

$$D = \frac{V_i^2 \sin 2\alpha}{g},$$

где $i = 1 \ldots n$;

$\alpha = [\alpha 1, \alpha 2]$;

$\Delta\alpha$ – шаг

Функция зависит от двух аргументов: V_i и α. Поэтому цикл вычислений сложный – цикл в цикле. Алгоритм будем разрабатывать в два этапа.

Шаг 1. Разработка укрупненной схемы алгоритма.

В алгоритме необходимо выполнить такие действия:

1) ввод исходных данных: **α1**, **α2**, **Δα** и массива **V**;

2) вычисление **n** значений **D** при заданных значениях **Vi,** где **i** = 1...n.

В алгоритме цикл реализуется с помощью конструкции **цикл с параметром** со счетчиком циклов i = 1...n. Алгоритм имеет структуру простого цикла (рис. 1.8, а).

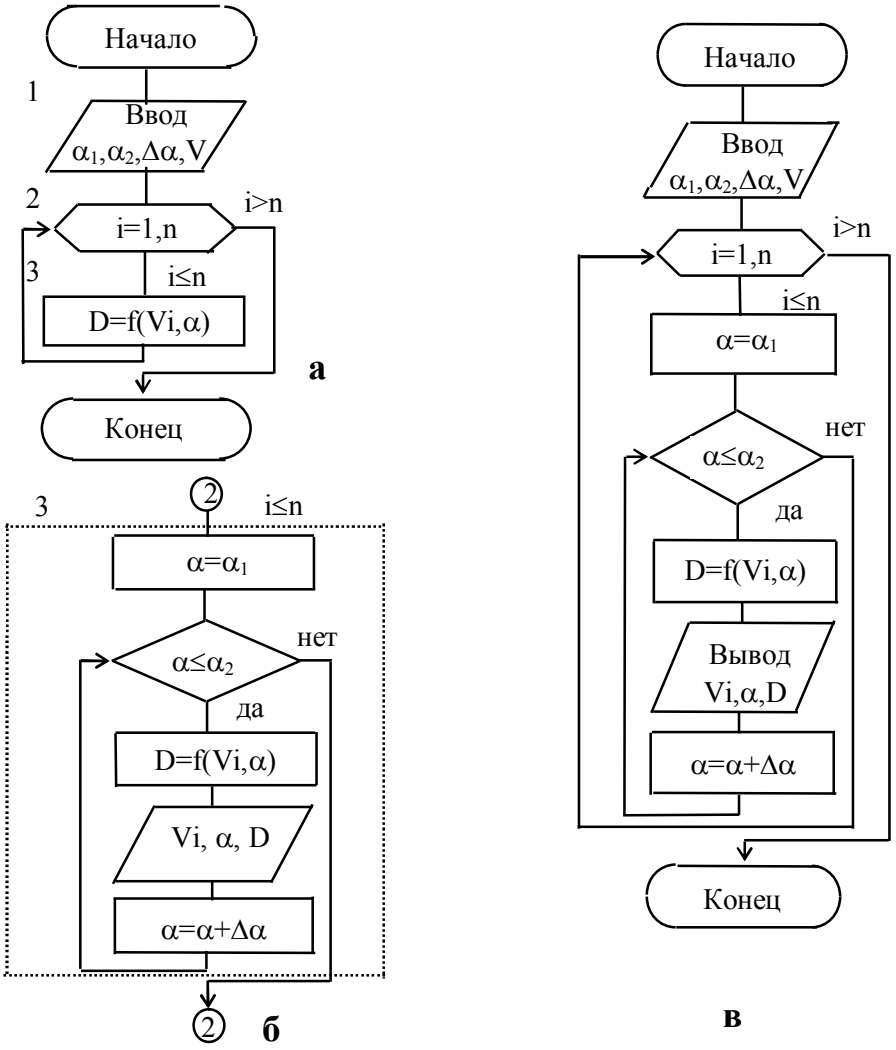


Рисунок 1.8 – Алгоритм с пошаговой детализацией:
а – укрупненная схема алгоритма; б – схема блока 3; в – схема алгоритма

Шаг 2. Детализация блока 3.

В блоке 3 необходимо выполнить следующие действия:

1) присвоить начальное значение переменной α: $\alpha = \alpha 1$;

2) организовать цикл вычисления и вывод значений функции $D=f(Vi, \alpha)$ при фиксированном значении V_i и при всех значениях α, которые изменяются от $\alpha 1$ к $\alpha 2$ с шагом $\Delta\alpha$.

Детализированная схема блока **3** приведена на рисунке 1.8, б.

После замещения блока **3** его детализированной схемой получим окончательную схему алгоритма, которая приведена на рисунке 1.8, в. Алгоритм имеет сложную циклическую структуру **цикл в цикле.**

Структурирование текста программы выполняется в соответствии с рекомендациями относительно стиля написания программ на алгоритмических языках:

3) ключевые слова разделов описания данных и составных операторов размещаются с одинаковым отступом вправо от начала строки;

4) операторы, которые входят в состав других операторов, сдвигаются вправо на одну или несколько позиций относительно начала основной конструкции;

5) каждое описание данных и оператор записывается в одной строке;

6) продолжение конструкций на новых строках сдвигаются вправо.

Благодаря этому достигается наглядность представления программы и легкость ее дальнейшего чтения и редактирования.

Использование вспомогательных алгоритмов

Решение сложных задач целесообразно сводить к решению более простых подзадач. На первом шаге задача разбивается на подзадачи, подзадачи – на еще меньшие подзадачи и так далее до элементарных задач, которые легко программируются. Определяются вспомогательные алгоритмы, их имена и разрабатывается основной алгоритм.

На втором и последующих шагах разрабатываются вспомогательные алгоритмы путем уточнения действий. Вспомогательный алгоритм реализует метод решения некоторой подзадачи или общий метод решения определенного класса подзадач. Например, алгоритм решения систем линейных уравнений, которое реализует метод Гаусса, может быть использован для решения прикладных задач, которые описываются системами линейных уравнений.

Математические формулы и выражения, которые описывают метод решения подзадачи, записываются с использованием абстрактных переменных. Переменные, которые используются в качестве аргументов вспомогательных алгоритмов, называются формальными параметрами. Заголовок алгоритма записывается в виде: **имя (список формальных параметров).**

Основной алгоритм является укрупненным алгоритмом решения задачи. Он включает указания на использование вспомогательных алгоритмов, которые сказываются с помощью символов **предопределенный процесс.**

В символах размещают команды. Каждая команда включает **имя вспомогательного алгоритма** и **список фактических параметров** (рис. 1.9). Фактическими параметрами могут быть имена переменных, числа и выражения, например, A(10, b, k+5). Количество, типы и порядок записи фактических и формальных параметров в списках должны совпадать.

Рисунок 1.9 – Блок вспомогательного алгоритма

Взаимодействие основного и вспомогательных алгоритмов рассмотрим на примере.

ПРИМЕР. Использование вспомогательного алгоритма. Вычислить расстояние до объектов с координатами (x1, y1), (x2, y2) от начала прямоугольных координат. Для решения задачи будем использовать вспомогательный алгоритм вычисления дальности по формуле:

$$r = \sqrt{x^2 + y^2},$$

где x, y – абстрактные переменные, которые являются аргументами алгоритма;

r – абстрактная переменная, которая обозначает расстояние до объекта.

Обозначим вспомогательный алгоритм как **dal (x, y)**,

где dal – имя алгоритма;

x, y – формальные параметры вспомогательного алгоритма.

Основной и вспомогательный алгоритмы решения задачи приведены на рисунке 1.10.

На момент исполнения вспомогательного алгоритма фактические параметры должны иметь конкретные значения, например x1 = 20, y1 = 50, x2 = 40, y2 = 30.

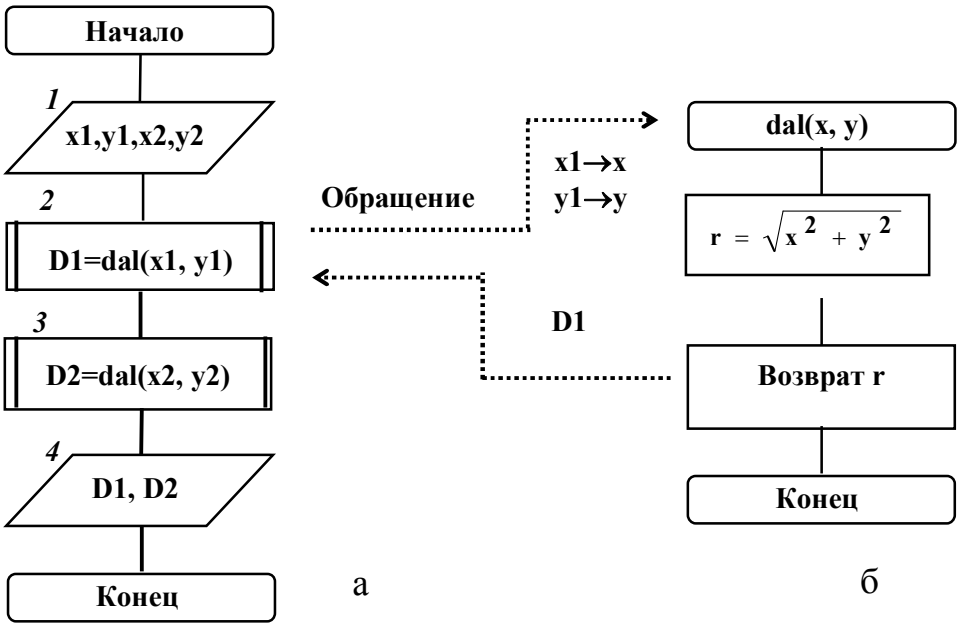


Рисунок 1.10 – Алгоритмы решения задачи:
а – основной алгоритм; б – вспомогательный алгоритм

При решении подзадачи фактические параметры заменяют соответствующие формальные параметры вспомогательного алгоритма и используются в вычислениях в качестве исходных данных. В данном примере вспомогательный алгоритм используется два раза (рис. 1.10 б): для вычисления расстояния D1 при x = x1, y = y1 и дальности D2 при x = x2, y = y2.

В программе вспомогательные алгоритмы реализуются функциями языка VBA. Функции являются подпрограммами, которые обеспечивают решение подзадач для конкретных исходных данных. Взаимодействие между программой и функциями осуществляется точно так же, как между основным и вспомогательным алгоритмами.

Вопросы для самоконтроля

1. Какие положения составляют основу структурного программирования?

2. Что является элементарной структурной единицей алгоритма и программы?

3. В чем заключается метод пошаговой детализации?

4. Какие способы конструирования алгоритмов вы знаете? Опишите каждый из них.

5. В чем заключается способ использования вспомогательных алгоритмов?

6. В чем заключается способ пошагового уточнения алгоритмов?

1.7 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объектно-ориентированное проектирование (ООП) – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированный анализ (ООА) – это методология, при использовании которой требования к проектируемой системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Как взаимодействуют ООА, ООД (объектно-ориентированный дизайн) и ООП? На результатах ООА формируются модели, на которых основывается ООД. В свою очередь, ООД создает фундамент для окончательной реализации компьютерной системы с использованием методологии ООП.

Наблюдение за действиями пользователя

Программное обеспечение, как и люди, имеет жизненный цикл. Под полным жизненным циклом программного обеспечения обычно понимают процесс его развития, состоящий из четырех последовательных фаз, которые называются:

- начало;
- исследование;
- проектирование;
- внедрение.

В свою очередь, фазы делятся на этапы. Разработка программного обеспечения – это процесс, охватывающий три первые фазы его жизненно-

го цикла. В ходе разработки постепенно создается программный продукт, готовый к внедрению. Разработка завершается тестированием программного комплекса и принятием решения о готовности программ к внедрению.

Программное обеспечение – это важная, но не единственная часть проектируемой компьютерной информационной системы. Другой ее частью является предметная область. Поэтому для разработки информационных систем, наряду с методом ООП, используются также метод объектно-ориентированного дизайна (или проектирования OOD) и метод объектно-ориентированного анализа. С помощью методов OOA и OOD моделируются предметные области информационных систем, которые иногда называют объектами автоматизации. Примером объекта автоматизации может служить, например, система подготовки необходимой части карты к печати в заданном масштабе.

В объектно-ориентированном программировании базовыми единицами программ и данных являются объекты. **Объект** – это сущность, которая четко проявляет свое поведение.

Объект состоит из следующих трех частей:

- имя объекта;
- состояние (переменные состояния);
- методы (операции).

Интерфейс объекта с его окружением определен полностью его методами, так как к его состоянию нет другого доступа извне, как через методы. Объект сохраняет свое состояние от обращения к обращению. Изменение состояний производится только через вызов методов этого объекта. Этим существенно ограничивается возможность введения объекта в недопустимое состояние или несанкционированное разрушение объекта. Возможность управлять состояниями объекта через вызов методов в конечном итоге будет определять поведение объекта.

Реализация методов (то есть, операций, выполняемых объектом), может быть задана различными способами. Объект может посылать сообщения другим объектам и принимать сообщения от них.

Сообщение является совокупностью данных определенного типа, передаваемых объектом-отправителем объекту-получателю, имя которого указывается в сообщении. Получатель реагирует или никак не реагирует на сообщение выполнением некоторой операции, имя которой также может быть указано в сообщении.

По своему смыслу объект является представителем некоторой реальной сущности (реального объекта, процесса или ситуации), которая:

- поддается хранению и обработке;
- способна воздействовать на другие объекты и вычислительную среду, посылая сообщения и реагировать на принимаемые сообщения.

Совокупность объектов в системе ООП образует среду, в которой выполняются вычисления путем обмена сообщениями между объектами. Состояние вычислительной среды в ООП оказывается разделенным на состояния объектов, это в принципе отличает объектно-ориентированные вычисления от вычислений, заданных на процедурно-ориентированных языках. Процедуры выполняются в общей памяти компьютера, в то время как объекты выполняют свои операции с учетом данных одного сообщения и своего собственного состояния.

Объекты с одинаковыми свойствами, то есть с одинаковыми наборами переменных состояния и методов, образуют класс. Каждый класс задается своим описанием на языке ООП. Описание включает информацию, необходимую для создания объектов данного класса и для их существования. Это информация о переменных состояния и операциях объекта.

Иначе говоря, объектно-ориентированный подход к программированию некоторой задачи состоит в том, чтобы создать некоторый инструментарий, присущий решаемой задаче, а затем уже программировать в терминах этой задачи.

Атрибуты объекта

Объект обладает атрибутами, определяющими его как самостоятельную сущность в мире программирования. У объектов, как и у любых компонентов **Visual Basic**, есть свойства, события и методы. Они отличаются от процедур тем, что поддерживают два принципа: наследование и инкапсуляцию.

Свойства объекта

Свойства описывают объект. Каждый объект, который вы используете или создаете в своей программе, обладает некоторым именем класса, по которому его можно отличить от других объектов.

Методы объекта

Методами называются действия, выполняемые объектом. Например, объект «окно» может отображать или скрывать себя на экране, а также из-

менять свои размеры.

Наследование

Наследованием называется способность объекта сохранять атрибуты класса-родителя. Например, созданный объект формы наследует свойства и методы своего класса-родителя. Он обладает такими свойствами, как **Name**, **WindowState** и **BorderStyle**, а также методами **Load**, **UnIoad** и **Hide.**

Чтобы создать экземпляр объекта в **Visual Basic**, следует указать на его принадлежность некоторому классу и затем воспользоваться ключевым словом **New**. Новый объект наследует свойства и методы родителя.

Инкапсуляция

Инкапсуляцией называется механизм, благодаря которому данные и методы объекта скрываются от внешнего мира. Программист изолируется от сложностей внутренней реализации объекта. Инкапсуляция составляет одну из самых сильных сторон ООП.

Инкапсуляция – это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект.

Внутри объекта данные и методы могут обладать различной степенью открытости (доступности). Они могут быть общедоступными или доступными только с помощью методов самого объекта. Обычно отрытые члены класса используются для того, чтобы обеспечить контролируемый интерфейс с его закрытой частью. Таким образом, комбинирование структуры данных с функциями (действиями или методами), предназначенными для манипулирования данными, называется инкапсуляцией.

Желательно при проектировании объекта стараться минимизировать открытые свойства. Закрывать все, с чем не должны работать внешние программы. Это предотвратит случайное внесение ошибок другими программистами.

Технология визуального программирования в интегрированной среде VISUAL BASIC

Разработка приложений в среде визуального программирования

включает следующие основные этапы.

1. Определение задач графического интерфейса на основе анализа сущности решаемой задачи, например:

- ввод исходных данных;
- вывод результатов вычислений;
- управление процессом вычислений: выбор алгоритма условий;
- настройка параметров отображения результатов.

2. Определение требуемого числа экранных форм, исходя из принципа «одна форма – одна задача графического интерфейса или группа совместных задач». Если объём исходных данных мал и объём результирующих данных невелик, то одна экранная форма может предназначаться для решения задачи ввода, вывода и управления процессом вычислений. Эти задачи графического интерфейса в данном случае являются совместимыми.

3. Выбор и размещение стандартных элементов на форме, которые необходимы для решения задачи графического интерфейса.

4. Установка исходных значений свойств каждого из элементов формы в окне свойств **Properties.**

5. Определение порядка взаимодействия пользователя программы с элементами экранной формы с целью выбора для каждого из них списка событий, на которые он должен реагировать, например: нажатие левой кнопки, правой кнопки, клавиш.

6. Написание программного кода обработчика события для каждого управляющего компонента, который будет выполняться при возникновении события.

Организация проекта

Проектом называется разрабатываемое приложение. Для формирования программы в среде VBA имеется ряд средств: конструктор форм, панель элементов, окна проекта, окно свойств и редактор кода.

Основная функция среды визуального программирования – это формирование заготовок (шаблонов программного кода) в соответствии со структурированной формой. Рассмотрим этот вопрос более подробно.

Программы строятся по модульному принципу и состоят из множества модулей. Принцип модульности важен для создания надежных, легко модифицируемых и сопровождаемых приложений. Все объекты компонен-

тов размещаются в объектах – формах.

Для каждой формы VBA создает отдельный модуль. В модулях осуществляется программирование задачи. В обработчиках событий объектов – форм и компонентов реализуются алгоритмы решения задачи. В основном они сводятся к обработке информации, содержащейся в свойствах одних объектов, и задании по результатам обработки свойств других объектов. При этом постоянно выполняется обращение к методам различных объектов.

Вопросы для самоконтроля

1. Объясните суть методологии объектно-ориентированного программирования (ООП).

2. Объясните суть методологии объектно-ориентированного анализа (ООА) и объектно-ориентированного дизайн (ООД).

3. Как соотносятся ООА, ООД и ООП?

4. Что такое объект и атрибуты объекта в ООП? Приведите примеры.

5. Поясните, что такое наследование и инкапсуляция в программировании.

6. Какие основные этапы включает разработка приложений в среде визуального программирования? Опишите эти этапы.

1.8 ПРИМЕНЕНИЕ UML

Общие сведения о UML

Язык моделирования – это нотация (система условных обозначений), в основном графическая, которая используется для описания проектов. Нотация представляет собой совокупность графических объектов, используемых в модели. Нотация является синтаксисом языка моделирования. Язык моделирования, с одной стороны, должен делать решения проектировщиков понятными пользователю, с другой стороны, предоставлять проектировщикам средства достаточно формализованного и однозначного определения проектных решений, которые реализуются при помощи программных комплексов, образующих целостную систему программного обеспечения.

Графическое изображение нередко оказывается наиболее емкой формой представления информации. При этом проектировщики должны учитывать, что графические методы документирования не могут полностью обеспечить декомпозицию проектных решений от постановки задачи

проектирования до реализации программ ЭВМ. Трудности возникают при переходе от этапа анализа системы к этапу проектирования и особенно к программированию.

Нотация представляет собой совокупность графических объектов, которые используются в моделях; она является синтаксисом языка моделирования. Например, нотация диаграммы классов определяет, каким образом представляются такие элементы и понятия, как класс, ассоциация и множественность.

К середине 90-х годов число методов моделирования сложных систем увеличилось более чем до 50-ти. В этой связи возникла проблема их обобщения и унификации. Частично она была решена в результате создания языка UML. По определению Гради Буча унифицированный язык моделирования (Unified Modelling Language, UML) является графическим языком для визуального представления, составления спецификаций, проектирования и документирования систем, в которых большая роль принадлежит программному обеспечению. В дальнейшем будем называть такие системы автоматизированными информационными системами (АИС) и полагать, что в работе АИС участвуют люди. С помощью языка UML можно разработать общесистемную документацию АИС, документацию ее программного обеспечения и создать многократно используемые (типовые) компоненты программного обеспечения.

Решающую роль в создании языка UML сыграли Гарди Буч, Джеймс Рамбо и Айвар Джекобсон. Они создали следующие методы моделирования различных сторон сложных систем:

- Метод Буча (Booch'93), ориентирован, в первую очередь, на моделирование программного обеспечения сложных систем;
- Метод Рамбо (OMT-2), ориентирован на анализ процессов обработки данных в информационных системах;
- Метод Джекобсона (метод OOSE), ориентирован на анализ требований к бизнес-приложениям.

Авторы этих методов объединились для создания унифицированного языка моделирования сложных систем, а также ими были сформулированы следующие требования к унифицированному языку, который был назван UML:

- Позволяет моделировать как программное обеспечение сложных систем, так и широкие классы самих систем и бизнес-приложений, с ис-

пользованием объектно-ориентированных понятий и методов;

- Обеспечивает взаимосвязь между базовыми понятиями моделей концептуального, программного и физического уровней;
- Понятен системным аналитикам и программистам;
- Поддерживается специальными инструментальными программными средствами, реализованными на различных компьютерных платформах.

В 1996 г. была создана первая версия языка UML. После этого ведущие компьютерные фирмы Microsoft, IBM, Oracle и многие другие осознали, что язык UML имеет стратегическое значение для их бизнеса. В результате был организован консорциум UML, деятельность которого оплачивается за счет ежегодных денежных взносов фирм членов консорциума.

Важную роль в создании языка UML сыграла его поддержка Группой по управлению объектами OMG (Object Management Group). Группа OMG объединяет около 300 ведущих компьютерных фирм. Она выпускает стандарты в области Интернет / Веб. Язык UML приобрел статус второго стратегического направления деятельности OMG. В 1997 г. были созданы версии языка UML 1.0 и 1.1. В 1998 г была создана версия UML 1.2, а в 1999 г. – версия UML 1.3. В настоящее время разработаны инструментальные программы поддержки языка UML. Наиболее известной из них является программы Rational Rose 2000 фирмы Rational Software. Кроме того, создан ряд средств визуального программирования, обеспечивающих прямую генерацию кода программ из UML моделей. Эти средства интегрированы с наиболее распространенными языками программирования Java, C++ и многими другими.

Классы и объекты UML

Классы

Существуют три категории классов, показываемых на UML диаграммах:

- абстрактные классы;
- создающие классы;
- реализующие классы.

Абстрактный класс (**abctract class**) не используется для создания объектов, а служит общей основой для подклассов. Например, класс линия, может быть абстрактным классом для классов первичная линия и вторичная линия.

Создающий класс (**creatable class**) представляет объекты, которые можно создавать непосредственно, используя синтаксис объявления объектов используемой среды разработки. В Visual Basic это действие выражается следующей синтаксической конструкцией:

Dim As New Object

Реализующий класс (**instantiable class**) не может напрямую создавать новые объекты, но объекты этого класса могут создаваться как свойство другого класса или функциями другого класса.

В браузере объектов VBA можно видеть все создающие и реализующие классы ArcGIS, но не абстрактные классы.

Отношения

Среди абстрактных, создающих и реализующих классов, возможны несколько видов отношений.

Ассоциации (**associations**) описывают отношения между классами. Ассоциация имеет определенную кратность с каждой из сторон. На рисунке 1.11 показано, что владелец может иметь один или несколько земельных участков, а земельный участок может принадлежать одному или нескольким владельцам.

Рисунок 1.11 – Отображение кратности в ассоциации

Кратность (**multiplicity**) представляет собой ограничение на число объектов, которые могут быть ассоциированы с другими объектами. Кратности обозначаются следующим образом:

- 1 – один и только один. Используется по умолчанию;
- 0..1 – ноль или один;
- M..N – от M до N (положительные целые числа);
- * или 0..* – от нуля до любого положительного числа;
- 1..* – от единицы до любого положительного числа.

Наследование (**type inheritance**) определяет специализированные классы, которые используют свойства и методы надкласса, а также имеют свои собственные дополнительные свойства и методы.

Схема (рис. 1.12) показывает, что классы *первичная линия* (создающий класс) и *вторичная линия* (создающий класс) являются потомками класса *линия* (абстрактный класс).

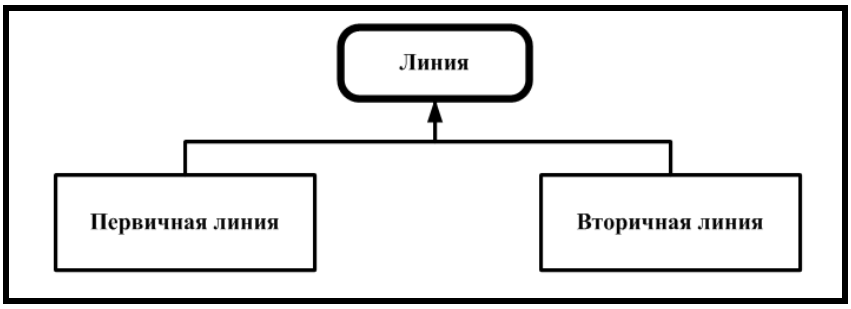


Рисунок 1.12 – Схема отображения кратности в ассоциации

Реализация (**instantiation**) определяет, что объект одного класса имеет метод, с помощью которого он может создать объект другого класса. Например (рис. 1.13), объект *опора* имеет метод для создания объекта *трансформатор*.

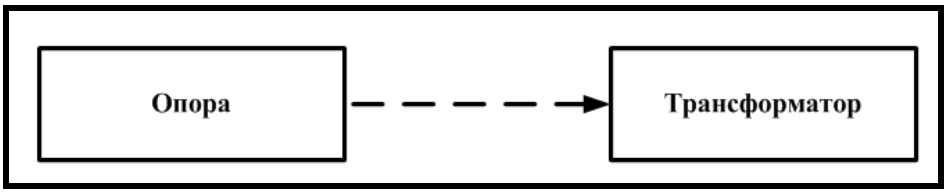


Рисунок 1.13 – Схема отображения реализации

Агрегирование (**aggregation**) является ассиметричной ассоциацией, в которой объект одного класса рассматривается как целое, состоящее из объектов другого класса, которые рассматриваются как части.

Трансформаторная группа содержит в своей конструкции ровно 3 трансформатора. *Трансформаторы* могут быть ассоциированы с *трансформаторной группой*, но они могут также существовать и после удаления объекта *трансформаторная группа* (рис. 1.14).

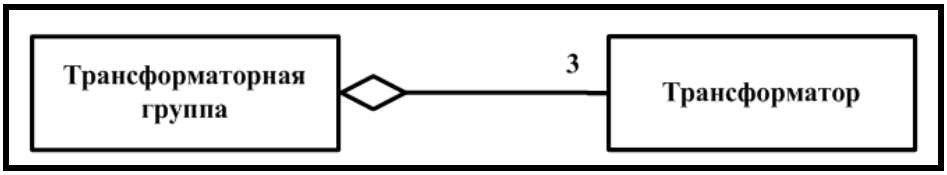


Рисунок 1.14 – Схема отображения агрегирования

Композиция (**composition**) является более сильной формой агрегирования, при которой объекты класса *целого* управляют существованием объекта класса *части*.

Например. *Опора* содержит одну или много *перекладин* (рис. 1.15). В данном случае, если удалить *опору*, то *перекладину* нельзя будет повторно использовать.

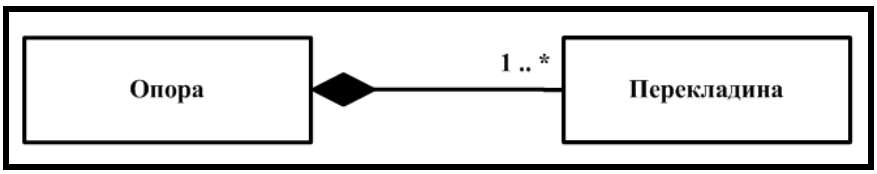


Рисунок 1.15 – Отображение композиции

Объект *опора* контролирует существование объекта *перекладина*.

Ключ к чтению UML-диаграмм

Диаграммы классов UML позволяют обозначать отношения между классами и их экземплярами. Они нужны, например, для моделирования прикладной области. Вначале необходимо выяснить, как относятся друг к другу классы в UML. На следующей структурной схеме (рис. 1.16) демонстрируются разновидности отношений.

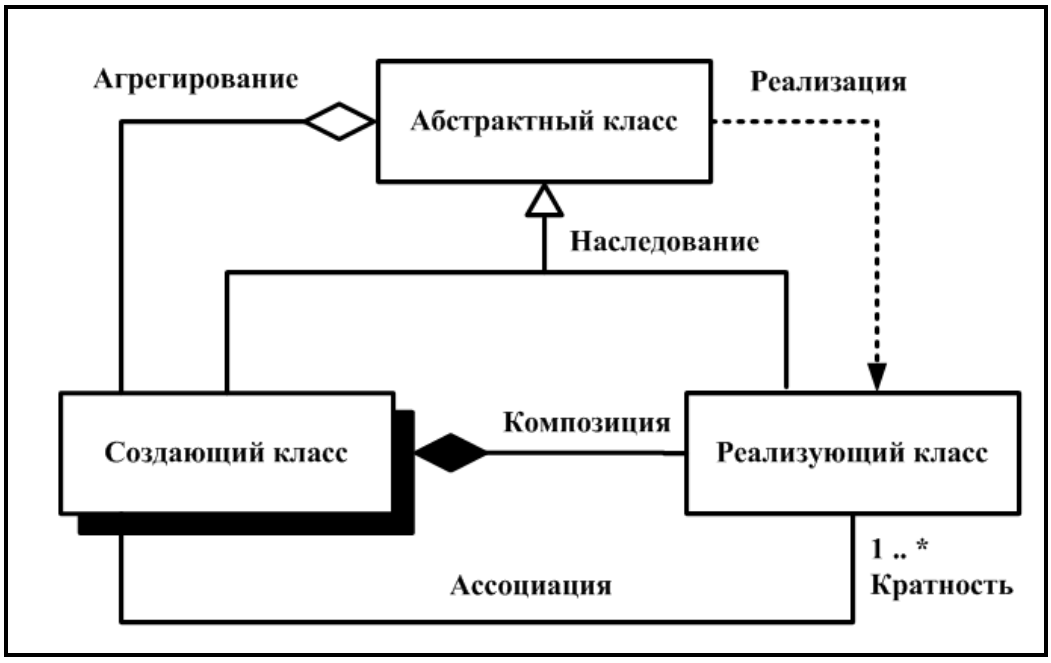


Рисунок 1.16 – Разновидности отношений в UML

Диаграммы UML

Диаграмма классов

Диаграммы классов являются центральным звеном объектно-ориентированных методов. Диаграмма классов определяет типы объектов системы и различного рода статические связи, которые существуют между ними. Имеются два основных вида статических связей:

- ассоциации (например, клиент может сделать заказ);
- подтипы (частный клиент является разновидностью клиента).

Построение диаграмм классов можно рассматривать в различных аспектах:

1. Концептуальный аспект – диаграммы классов отображают понятия изучаемой предметной области (моделируемой организации). Эти понятия, естественно, будут соответствовать реализующим их классам, однако такое прямое соответствие зачастую отсутствует. На самом деле концептуальная модель может иметь весьма слабое отношение или вообще не иметь никакого отношения к реализующему ее программному обеспечению, поэтому ее можно рассматривать как не зависимую от средств реализации (языка программирования).

2. Аспект спецификации – модель спускается на уровень программного обеспечения (ПО), но рассматриваются только интерфейсы, а не программная реализация классов (под интерфейсом здесь понимается набор операций класса, видимых извне).

3. Аспект реализации – модель действительно определяет реализацию классов ПО. Этот аспект наиболее важен для программистов.

При построении диаграммы необходимо выбрать единственный аспект. При чтении диаграммы следует выяснить, в соответствии с каким аспектом она строилась.

На рисунке 1.17 изображена простая модель классов, связанная с обработкой заказов клиентов.

При описании каждого фрагмента модели и рассмотрени его возможной интерпретации с различных точек зрения видно, что ассоциации представляют собой связи между экземплярами классов (личность работает в компании, компания имеет ряд офисов).

С концептуальной точки зрения ассоциации представляют собой концептуальные связи между классами. На диаграмме показано, что **Заказ**

должен поступить от единственного **Клиента,** а **Клиент** в течении некоторого времени может сделать несколько **Заказов.** Каждый из этих **Заказов** содержит несколько **Строк заказа,** каждая из которых соответствует единственному **Продукту.**

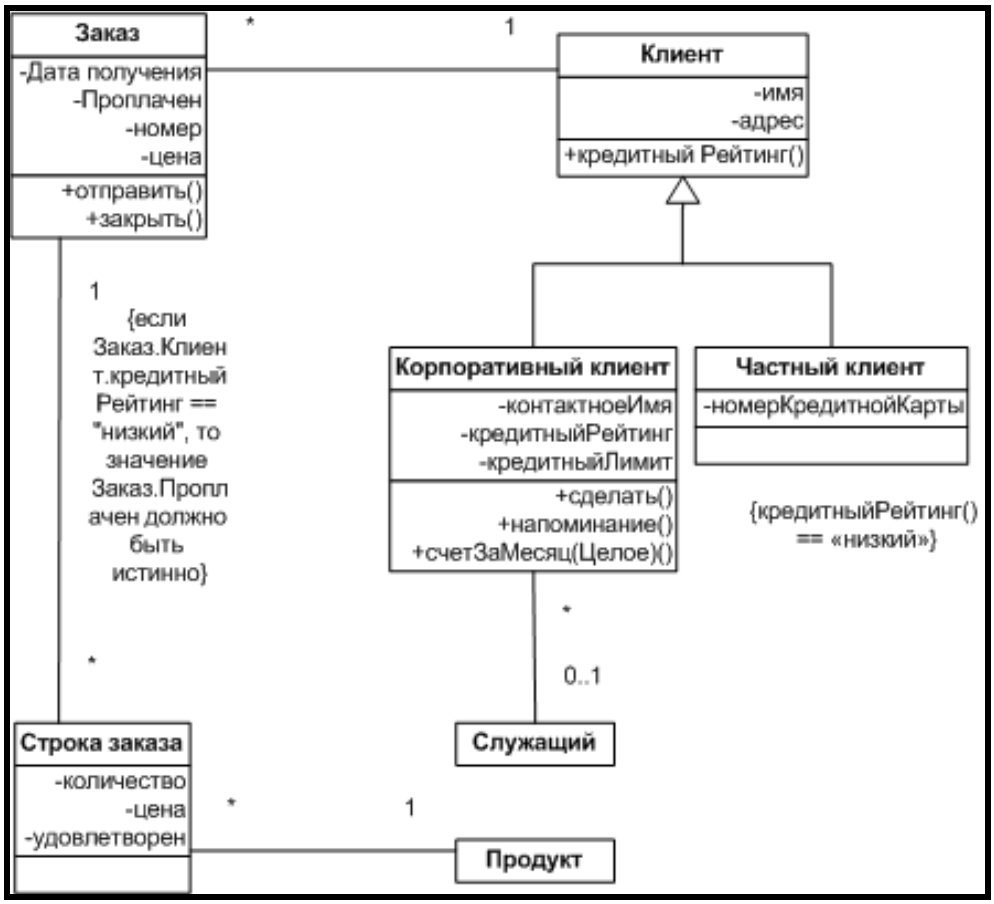


Рисунок 1.17 – Диаграмма классов процесса обработки заказов клиентов

Каждая ассоциация обладает двумя ролями. Каждая роль представляет собой направление ассоциации. Таким образом, ассоциация между **Клиентом** и **Заказом** содержит две роли: одна от **Клиента** к **Заказу,** другая – от **Заказа** к **Клиенту.**

Роль может быть явно поименованна с помощью метки. Например, роль ассоциации в направлении от **Заказа** к **Строкам заказа** называется «**позиция заказа**». Если такая метка отсутствует, роли присваивается имя класса цели. Таким образом, роль ассоциации от **Заказа** к **Клиенту** может быть названа **Клиент** (термины «**начало**» (source) и «**цель**» (target) употребляются для обозначения классов, являющихся соответственно начальным и конечным для ассоциации).

Диаграмма взаимодействия

Диаграммы взаимодействия (**interaction diagrams**) являются моделями, описывающими поведение взаимодействующих групп объектов.

Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображаются ряд объектов и те сообщения, которыми они обмениваются между собой.

Зачастую на этапе спецификации требований необходимо показать не только алгоритм действий или изменение состояния объекта, но и обмен сообщениями между отдельными объектами системы. Данную задачу решают диаграммы взаимодействия. Диаграмма взаимодействия предназначена для моделирования отношений между объектами (ролями, классами, компонентами) системы в рамках одного прецедента.

Данный вид диаграмм отражает следующие аспекты проектируемой системы:

- обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними системами);
- ограничения, накладываемые на взаимодействие объектов;
- события, инициирующие взаимодействия объектов.

В отличие от диаграммы деятельности, которая показывает только последовательность (алгоритм) работы системы, диаграммы взаимодействия акцентируют внимание разработчиков на сообщениях, инициирующих вызов определенных операций объекта (класса) или являющихся результатом выполнения операции.

Расширение нотации диаграмм взаимодействия в UML 2.0 позволяет аналитикам на более детальном уровне проработки требований по возможности заменять диаграммы деятельности диаграммами взаимодействия. Таким образом, основной целевой аудиторией для диаграммы взаимодействия будет команда разработчиков. Для заказчика данный вид диаграмм будет интересен только в рамках моделирования взаимодействия проектируемой инфорационной системы и сторонних систем, работающих на стороне заказчика.

Для описания взаимодействия объектов в UML предусмотрены следующие виды диаграмм:

- Диаграмма последовательности – моделирует последовательность обмена сообщениями между объектами;

- Диаграмма коммуникаций – модулирует структуру взаимодействующих компонентов (для данного вида диаграммы в UML используется наименование «диаграмма кооперацій»);
- Временные диаграммы – моделирует изменение состояния нескольких объектов в момент взаимодействия;
- Диаграмма обзора взаимодействия – сочетание диаграммы деятельности и диаграммы последовательности.

На диаграмме последовательности объект изображается в виде прямоугольника на вершине пунктирной вертикальной линии (рис. 1.18). Эта вертикальная линия называется линией жизни (**lifeline**) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия. Такую форму представления впервые ввел Ивар Якобсон.

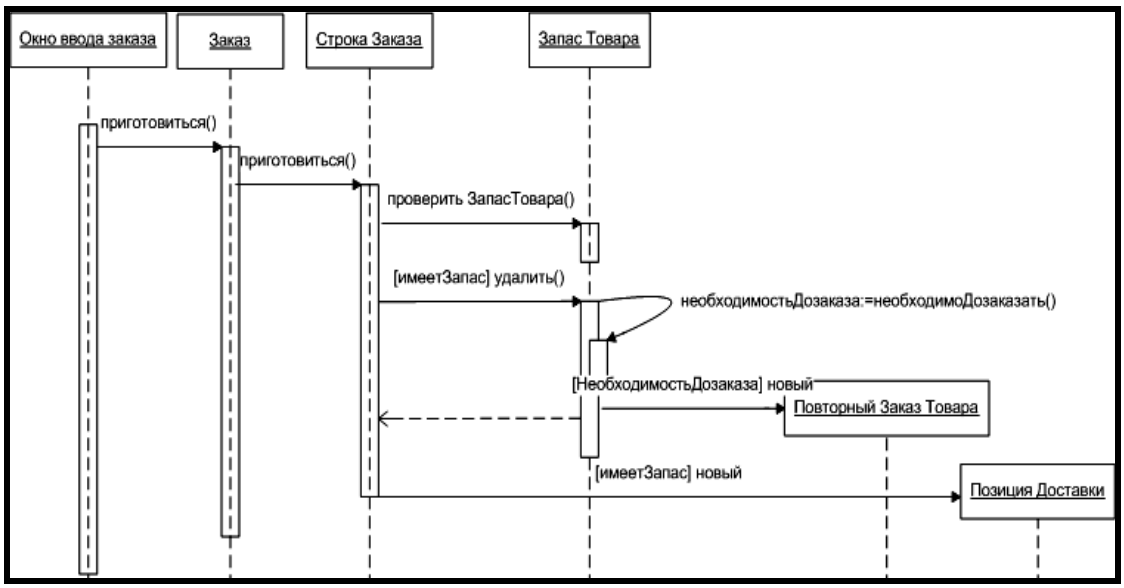


Рисунок 1.18 – Диаграмма последовательности

Диаграмма описывает следующее поведение:

- Окно **Ввода Заказа** посылает **Заказу** сообщение «приготовиться».
- Заказ посылает данное сообщение каждой **Строке заказа** в данном **Заказе**.
- Каждая **Строка заказа** проверяет состояние определенного **Запаса товара**.
- Если данная проверка удовлетворяется (результат – true), то **Строка заказа** удаляет соответствующее количество товара из **Запаса**.
- В противном случае количество **Запаса** снижается до уровня повторного заказа, и **Запас** запрашивает новую поставку товара.

Использование объектно-ориентированного подхода

В качестве предметной области рассмотрим работу подразделения учета налогоплательщиков-организаций. На начальной стадии (или стадии формирования требований) строится начальная диаграмма вариантов использования (рис. 1.19).

При построении диаграммы вариантов использования в первую очередь составляется список всех основных действующих лиц (физических лиц или внешних систем, которые будут взаимодействовать с создаваемой системой). Их можно идентифицировать, задавая следующие вопросы:

- кто использует систему непосредственно?
- кто отвечает за эксплуатацию системы?
- какое внешнее оборудование используется системой?
- какие другие системы взаимодействуют с данной системой?

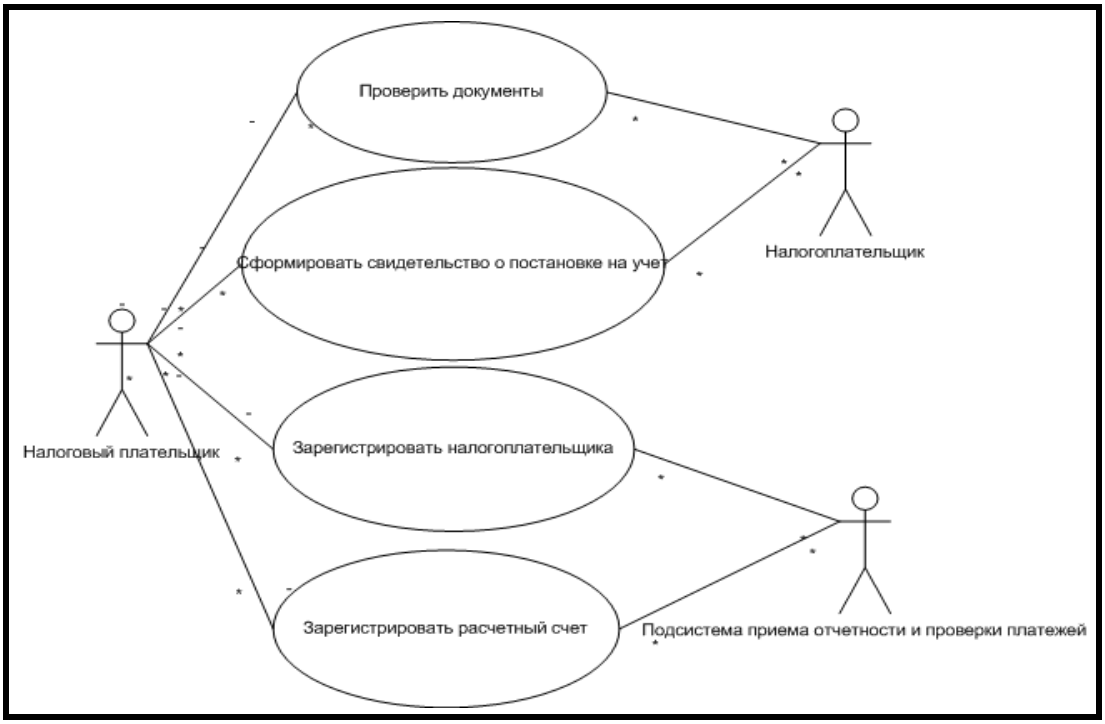


Рисунок 1.19 – Начальная диаграмма вариантов использования

Варианты использования идентифицируются исходя из следующих соображений: каждый вариант использования представляет собой некоторую функцию, выполняемую системой в ответ на воздействие действующего лица, и характеризует конкретный способ применения системы, диалог между действующим лицом и системой. Нужно также иметь в виду, что впоследствии варианты использования будут служить для описания

требований к системе, общения с конечными пользователями и экспериментами предметной области, а также для тестирования системы.

На стадии проектирования уточняется диаграмма вариантов использования и строится архитектура системы, основой которой являются диаграммы классов. В данном примере ограничимся построением диаграммы классов и диаграммы взаимодействия. Диаграммы взаимодействия строятся для уточнения диаграммы вариантов использования и перехода к диаграммам классов. Так, диаграмма последовательности (рис. 1.20) иллюстрирует один из возможных сценариев развития событий в рамках варианта использования «Зарегистрировать налогоплательщика». Предполагается, что налогоплательщик ставится на учет впервые и все его документы в полном порядке.

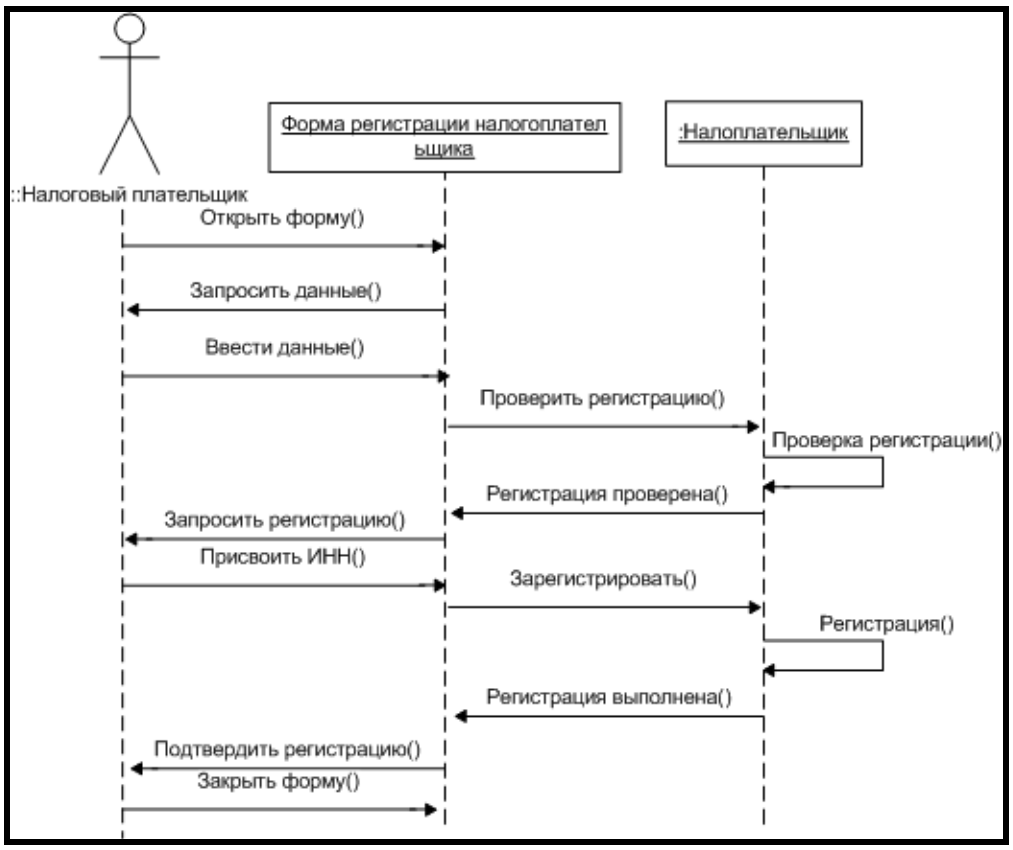


Рисунок 1.20 – Диаграмма последовательности для варианта использования «Зарегистрировать налогоплательщика»

Структура программной системы описывается с помощью нескольких диаграмм классов, главная из которых представляет собой диаграмму пакетов, а остальные диаграммы раскрывают содержимое каждого из пакетов.

При построении диаграммы классов предметной области выделение этих классов (классов-сущностей) может быть аналогично выделению сущностей в процессе моделирования данных. Данные классы должны иметь концептуальный характер и отвечать на вопрос «что?», а не «как?».

Начальный список может быть составлен следующим образом:

- в описании исходных данных выделяются кандидаты в классы-существительные, которые потенциально могут соответствовать классам (при этом следует помнить, что существительные могут также относиться к объектам, ассоциациям или атрибутам классов);
- анализируются роли кандидатов в системе. Каждый класс должен выполнять некоторые действия и взаимодействовать с другими классами. Каждый класс должен иметь уникальное имя, отражающее характер абстракции, представляемой данным классом. Если классу трудно придумать краткое и содержательное имя, то это является характерным признаком неудачного выделения класса. Рассматривается каждая возможная пара классов и устанавливается существование ассоциации между ними (по аналогии с установлением связей между сущностями в процессе моделирования данных). Присваиваются наименования ролям ассоциаций, и определяется их множественность.

Далее составляется список атрибутов каждого класса (по аналогии с определением атрибутов сущностей при моделировании данных). Процесс определения атрибутов должен быть непродолжительным, поскольку существенные атрибуты могут быть добавлены впоследствии. При этом следует убедиться, что не пропущены существенные характеристики, представленные в исходных данных.

Определяются действия (операции), выполняемые каждым классом. При определении операций нужно учитывать следующие рекомендации:

- каждая операция должна выполнять одну простую функцию;
- название операции должно отражать результат функции, а не то, как она выполняется.

Примерами простых операций могут быть:

- получить значение атрибута;
- установить значение атрибута;
- добавить или исключить связь с другим объектом;
- удалить данный объект.

Полученная в результате диаграмма классов предметной области показана на рисунке 1.21.

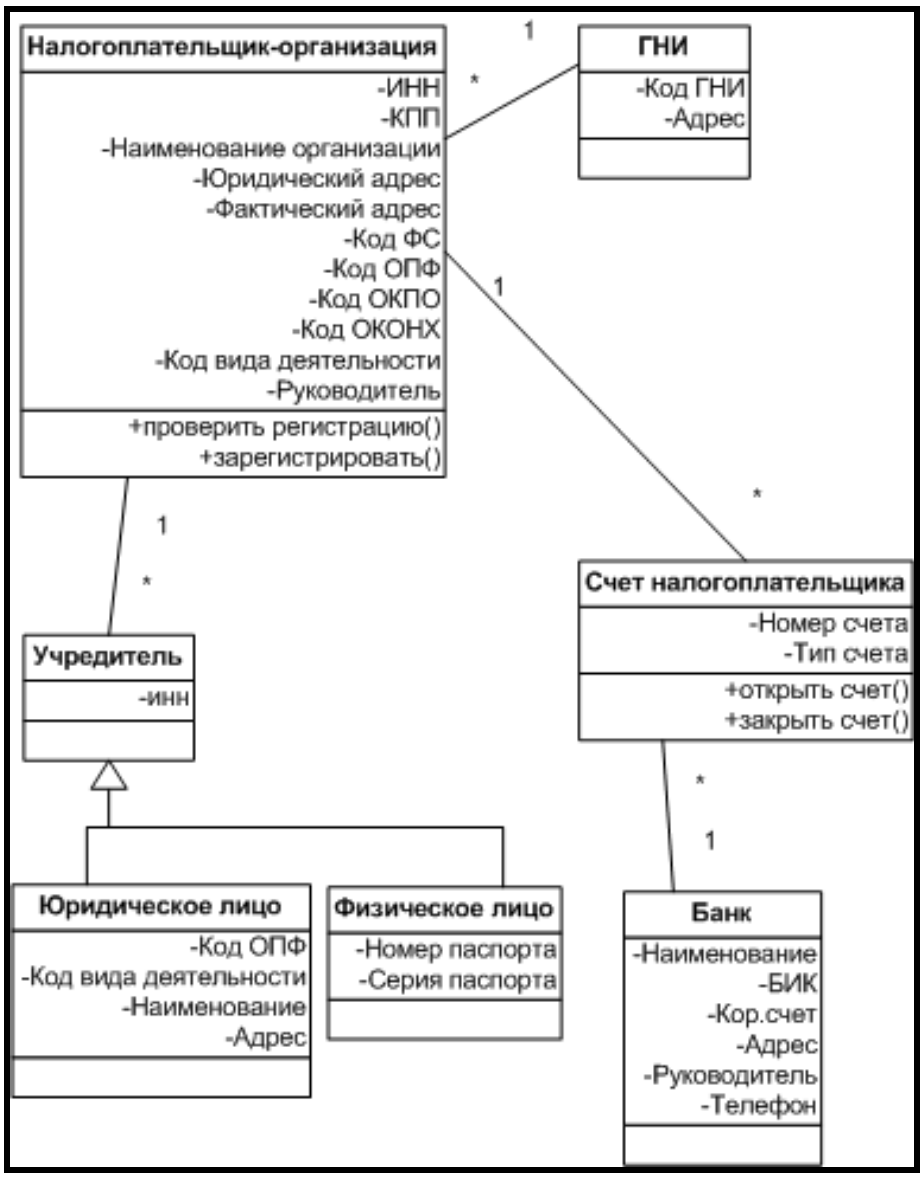


Рисунок 1.21 – Диаграмма классов предметной области

Механизм пакетов

Один из подходов к решению проблемы разработки базы геоданных заключается в группировке классов в компоненты более высокого уровня. Эта идея проявляется во многих объектно-ориентированных методах. В UML такой способ группировки носит название **механизма пакетов** (**package**).

Механизм пакетов применим к любым элементам модели, а не только к классам. Если для группировки классов не использовать некоторые эвристики, то она становится весьма произвольной. Одна из них, которая в

основном используется в UML – это зависимость. Таким образом, диаграмма пакетов представляет собой диаграмму, содержащую пакеты классов и зависимости между ними (рис. 1.22). Пакеты и зависимости являются элементами диаграммы классов, получается, что диаграмма пакетов – это всего лишь форма диаграммы классов.

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

В идеальном случае, только изменения в интерфейсе класса должны воздействовать на другие классы. Искусство проектирования больших систем включает в себя минимизацию зависимостей – таким способом снижается воздействие изменений и требуется меньше усилий на их внесение.

На рисунке 1.22 показаны классы предметной области, сгруппированные в два пакета: **Заказы** и **Клиенты.** Оба пакета, в свою очередь, являются частью общего пакета предметной области. Приложение **Сбора Заказов** имеет зависимости с обоими пакетами предметной области. Пользовательский интерфейс **Сбора Заказов** имеет зависимости с **Приложением Сбора Заказов** и AWT (средством разработки графического интерфейса пользователя (GUI) в языке Java).

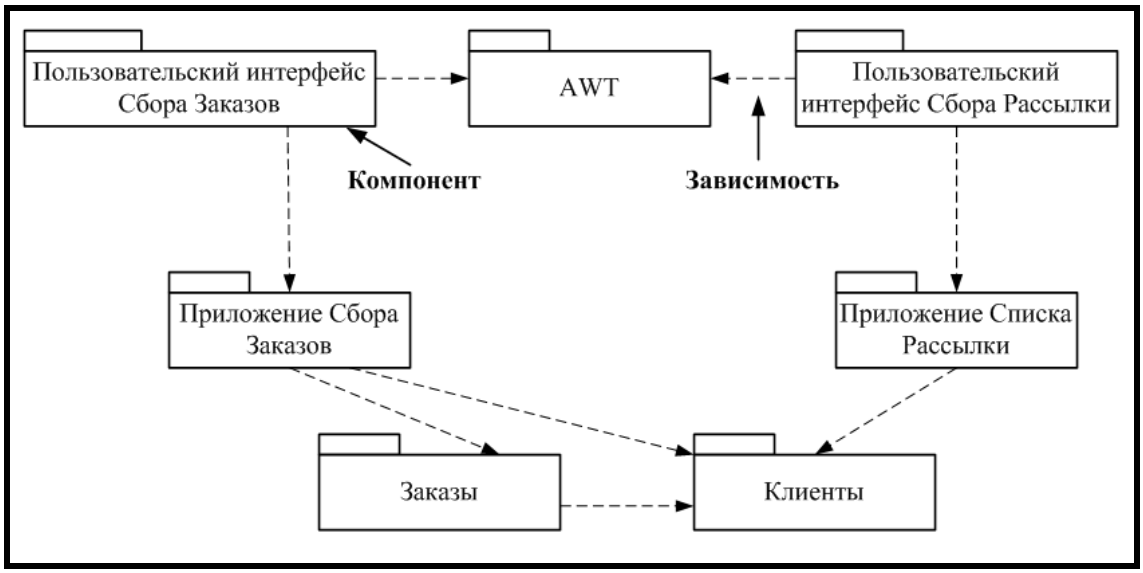


Рисунок 1.22 – Классы предметной области, сгруппированные в два пакета: Заказы и Клиенты

Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость. Например, если любой класс в пакете **Список Рассылки** зависит от какого-либо класса в пакете **Клиенты,** то между соответствующими пакетами существует зависимость.

Существует очевидное сходство между зависимостями пакетов и составными зависимостями. Однако между ними имеется принципиальное различие: зависимости пакетов не являются транзитивными. Чтобы понять, почему это так важно для зависимостей, обратимся к рисунку 1.16. Если какой-либо класс в пакете **Заказы** изменяется, то это совсем не означает, что должен измениться пакет **Пользовательский интерфейс Сбора Заказов.** Это означает всего лишь, что может измениться пакет **Приложение Сбора Заказов.** Пакет **Пользовательский интерфейс Сбора Заказов** должен измениться только в том случае, если меняется интерфейс пакета **Приложение Сбора Заказов.** В данной ситуации **Приложение Сбора Заказов** защищает **Пользовательский интерфейс Сбора Заказов** от изменений в заказах.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в системе, однако они помогают выделить эти зависимости, а после того, как они все окажутся на виду, остается только поработать над снижением их количества. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

Вопросы для самоконтроля

1. Для чего используется язык UML? Как расшифровывается эта аббревиатура?
2. Что такое **Нотация**, из чего она состоит?
3. Какие задачи можно описывать с помощью языка UML?
4. Какие категории классов можно отображать на UML диаграммах? Опишите каждый из этих классов.
5. Какие виды отношений возможны между классами?
6. Что такое **Кратность** и для каких целей она используется?
7. Что такое **Наследование** и для каких целей оно используется?
8. Что такое **Реализация** и для каких целей она используется?
9. Для чего используется **Агрегирование** и **Композиция** в UML?

10. Для описания каких процессов используются диаграммы классов в UML?

11. Для описания каких процессов используются диаграммы взаимодействия в UML?

12. В чем суть способа группировки под названием **Механизма пакетов**?

1.9 ПРИНЦИПЫ ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ

Экстремальное программирование – молодая, но быстро развивающаяся методология разработки программного обеспечения. Она получила признание и широкое распространение благодаря ориентации на обычных людей, максимальному упрощению бюрократических процедур и обилию качественной литературы, освещающей эту методологию.

С чего начинается экстремальное программирование? С понимания того, что разработчик программного обеспечения старается максимально снизить стоимость разработки. А для этого необходимо интенсивно сотрудничать с заказчиком, понимать его интересы и, в конце концов, сделать именно то, чего он хочет: не больше и не меньше.

В основе экстремального программирования лежат четыре базовых принципа: общение, простота, обратная связь и храбрость. Именно с них необходимо начинать.

Экстремальное программирование предлагает готовое решение: делайте все максимально просто, держите постоянный контакт с заказчиком, позвольте ему активно следить за процессом разработки, приветствуйте изменения – и успех практически обеспечен.

Простота программного кода и общение с заказчиком

В командах, работающих по методу экстремального программирования, всегда приветствуется общение – самое быстрое средство обмена информацией и опытом. Это очень важно, когда требуется максимальная скорость разработки. Но общение, как и любое другое полезное начинание, требует постоянной поддержки. Именно поэтому кто-то из команды должен взять на себя ответственность следить за общением. Общение и необходимость объяснения своих действий другим членам команды вынуждает делать все максимально просто. Главная цель – максимальная понятность

кода для других разработчиков. Если это не получается с первого раза, над упрощением работают еще и еще, пока не будет достигнута главная цель. Если от этой цели отклоняются, то действия корректируются соответствующим образом. В процессе разработки программ часто разрабатывается нечто, результат чего не виден. Поэтому в экстремальном программировании необходимо обеспечить максимально быструю обратную связь.

Храбрость при проектировании программного кода

Храбрость очень важна в работе. Разве можно без храбрости принять на себя ответственность за выполнение какой-то задачи, да еще в конкретные сроки? Разве можно без храбрости разработчику признать свою ошибку в оценке задачи и вовремя предупредить об этом остальных вместо того, чтобы поставить их перед фактом уже тогда, когда все сроки истекут? Польза храбрости налицо, и каждый успех, даже в самой маленькой задаче, способен эту храбрость развить.

Методики экстремального программирования

Игра в планирование

Мир слишком изменчив и непредсказуем, чтобы полагаться на постоянство ситуации. То же происходит и при разработке программного обеспечения: о редкой системе можно сказать, что ее окончательный вид был заранее известен в деталях еще в самом начале разработки. Обычно заказчик постоянно старается что-то поменять, что-то улучшить, а что-то вообще выбросить из системы. Это и есть изменчивость требований. К счастью, разработчику дано умение прогнозировать возможные варианты и, таким образом, держать ситуацию под контролем.

В экстремальном программировании планирование – неотъемлемая часть разработки. То, что планы могут поменяться, учитывается с самого начала. Методикой, которая позволяет прогнозировать ситуацию и безболезненно мириться с изменениями, является игра в планирование. В ходе такой игры можно быстро собрать известные требования к системе, оценить и запланировать их разработку в соответствии с приоритетностью.

Как и любая другая игра, планирование имеет своих участников и свою цель. Ключевой фигурой является, конечно же, заказчик. Именно он сообщает о необходимости той или иной функциональности. Программи-

сты же дают ориентировочную оценку каждой функциональности. Достоинство игры в планирование заключается в единстве цели и солидарности разработчика и заказчика. В случае победы побеждают все, в случае поражения все проигрывают. Но при этом каждый участник идет к победе своей дорогой: заказчик выбирает наиболее важные задачи в соответствии с бюджетом, а программист оценивает задачи в соответствии со своими возможностями по их реализации. Экстремальное программирование предполагает, что разработчики в состоянии сами решить, за какой промежуток времени они справятся со своими задачами.

В идеальной ситуации игра в планирование с привлечением заказчика и программиста должна проводиться каждые 3 – 6 недель, до начала следующей итерации разработки. Это позволяет довольно просто внести коррективы в соответствии с успехами и неудачами предыдущей итерации.

Тестирование до начала разработки

Обычно для тестирования нанимают специально обученного человека, который периодически выполняет одни и те же действия. В экстремальном программировании роль тестирования иная – вначале идет тест, а потом код. Как же тестировать то, чего еще нет? Ответ прост: необходимо тестировать идеи – чего следует ожидать от будущей части программы. Это позволит лучше понять, что требуется сделать программистам, и проверить работоспособность кода сразу, как только он будет написан. Опыт показывает, что такой подход не только не замедляет, но и ускоряет разработку. Ведь знание того, что нужно сделать, и требуемого объема работ позволят сэкономить время, отказавшись от реализации невостребованных в данный момент деталей.

Парное программирование

Опытные разработчики заметили, что периодический просмотр чужого кода положительно влияет на его качество. Мастера экстремального программирования развили этот подход: в ходе разработки код пересматривается постоянно посредством приема, именуемого парным программированием.

Парное программирование заключается в следующем. Два программиста работают за одним компьютером, пользуясь общими – клавиатурой и мышкой. Такой подход не только повышает качество кода, но и обеспечивает более тесное общение, неявную передачу знаний, более эффектив-

ное использование рабочего времени, позволяет выявлять глобальные ошибки на ранних стадиях.

Исследования парного программирования показывают, что затраты на разработку не увеличиваются вдвое, а за счет экономии времени остаются приблизительно на том же уровне.

Постоянная переработка

Добавление каждой новой функциональности и разрастание кода усложняют разработку, повышает количество ошибок и соответственно, внесение последующих изменений. Одним из преимуществ экстремального программирования является компенсация добавления функциональности – усовершенствование кода. Это и есть переработка кода, или рефакторинг.

Правила хорошего тона для большинства языков программирования, особенно объектно-ориентированных, давно сформулированы. Если ими грамотно руководствоваться при написании нового кода и учитывать все возможные изменения, то никакой переработки не потребуется. Но такое маловероятно. Код не всегда получается корректным с первого раза, отчего трудозатраты стремительно растут.

Переработка кода позволяет адекватно и немедленно реагировать на каждое изменение.

Простота разработки

Экстремальное программирование учит все делать максимально просто – не усложняя себе работу, в том числе и при разработке программного обеспечения. Простую программу легче поддерживать, в нее легче вносить изменения, она менее подвержена ошибкам.

Коллективное владение кодом

Традиционно программный продукт поделен на сферы влияния между несколькими разработчиками. Каждый программист разрабатывает и отвечает за определенные участки программы. В такой схеме каждый разработчик является специалистом узкого профиля. Если потребуется внести изменения в «чужой» код, программист вынужден будет попросить об этом «владельца» соответствующей части программы.

Такая расстановка сил оправдана, если в команде царит абсолютное доверие и надежность каждого безупречна. Но что произойдет, если один из разработчиков уйдет в отпуск, заболеет или уволится? Для того чтобы

компенсировать возможные «выпадающие звенья», используется принцип коллективного владения кодом. Каждым участком кода должны владеть как минимум два программиста, и любой член команды может внести изменения в любую часть кода.

Работоспособность такого подхода поддерживается другими методиками, такими как модульное тестирование и простота разработки. Внести изменения в простой код несложно, но если после этого нарушится что-то написанное ранее, об этом сразу просигнализируют модульные тесты.

Продолжающаяся интеграция

Очень часто разработчики сталкиваются с проблемами поздней интеграции, когда новая функциональность оказывается несовместима с остальным проектом. Единственным эффективным средством решить такую проблему является продолжающаяся интеграция. Интегрировав работоспособный участок кода раньше, можно побороть или даже предотвратить несовместимость на ранней стадии проекта.

Несмотря на свою простоту, такая методика имеет свои правила использования, такие как успешность выполнения имеющихся модульных тестов для интегрируемой функциональности, наличие функциональных или приемочных тестов и, конечно же, возможность отката к предыдущему состоянию. Как правило, интеграция и разрешение сопутствующих трудностей выполняются на отдельном компьютере несколькими программистами. Это позволяет свести к минимуму риск нежелательных последствий интеграции.

Участие заказчика в процессе разработки

Основной проблемой разработки программного обеспечения является недостаток знаний программистов в разрабатываемой предметной области. Экстремальное программирование нашло выход и из этой ситуации. Это не стажировка разработчика на предприятии заказчика, это участие заказчика в процессе разработки.

Разве может программист, досконально не понимая суть вопроса угадать, чего хочет заказчик? Ответ очевиден. Самым простым способом преодолеть такое неудобство – задать заказчику прямой вопрос. В определенных случаях это оправдано, хотя и дороже обходится. Реальный опыт ведения прикладных проектов показывает, что невозможно собрать все требования заранее. Более того, даже если предположить, что все требова-

ния на текущий момент собраны, все равно останется одно узкое место: программы, как и все в природе, не создаются мгновенно, а тем временем бизнес-процессы могут измениться. Это следует учитывать.

Многие сомневаются в возможности привлечения заказчика к процессу разработки. Действительно, заказчики бывают разные. Если привлечь заказчика или его представителя не удается, иногда оказывается целесообразным временный наем специалиста в разрабатываемой области. Такой шаг сократит неясности в работе, повысит скорость разработки и приблизит проект к тому, что желает получить заказчик. Это может быть выгодно и с финансовой стороны: ведь оплата труда программиста порой значительно превышает оплату специалистов других отраслей.

Быстрый выпуск версий

Ни один аналитик, даже самый квалифицированный, не поймет потребности заказчика лучше, чем он сам, пользуясь готовым продуктом некоторое время. Эта ценная особенность человеческой природы легла в основу новой методики, которая формулируется следующим образом: дайте заказчику определиться со своими потребностями раньше, выпустив версию быстрее.

Впрочем, у этого подхода есть определенные ограничения: во-первых, пользователи не всегда готовы к быстрым переменам, а во-вторых, выпускаемый продукт должен быть полностью протестирован. Если с первой проблемой справиться проблематично, то вторая успешно решается применением автоматизированных тестов, модульных или приемочных.

Сорокачасовая рабочая неделя

Экстремальное программирование категорически против нарушения принятых норм трудового права. Это продиктовано не только соображениями законности и гуманности, а – в первую очередь – необходимостью повышения эффективности работы и строгой организации. Ведь экстремальное программирование – коллективная работа, рассчитанная не на индивидуумов, а на всю группу целиком. А такая работа, как например, парное программирование, возможна лишь при синхронизации рабочего распорядка ее участников. И самое главное, человеку чтобы сохранить здоровье и работоспособность, необходим полноценный отдых. Во многих западных фирмах поздний уход с работы расценивается как неуспеваемость или неспособность правильно распорядиться своим рабочим временем.

В большинстве случаев это так и есть. Да и с медицинской точки зрения, задержки на работе ведут к постоянной усталости, раздражительности и снижению активной мозговой деятельности. Нормы есть нормы, и их стоит придерживаться.

Стандарты кодирования

Экстремальное программирование в качестве обязательного условия успешности работы над проектом выдвигает требование применения стандартов кодирования. Это оправдано с точки зрения многих позиций: единообразный код более понятен другим разработчикам.

Метафора системы

Экстремальное программирование предполагает еще один эффективный инструмент для освоения разрабатываемой системы. В частности, первое представление о системе может быть получено посредством метафоры, или сравнения с существующими аналогами.

Человеческое мышление построено на образах: каждое слово вызывает в памяти ассоциацию с характерным образом. Специалисты учли этот факт и выделили его в отдельную методику. Теперь для понимания системы используется сравнение наиболее похожего ранее известного продукта или предмета с разрабатываемым.

Преимущество экстремального программирования

Эти методики собраны воедино не случайно. Их непротиворечивая совокупность способна ввести процесс разработки в интеллектуальный резонанс, заметно повысив качество продукта и приблизив время его выпуска. Основное преимущество экстремального программирования – прогнозируемость и сведение к минимуму затрат на разработку; предоставление заказчику того продукта, который он желает получить на момент выпуска; и конечно же общение и обучение разработчиков без отрыва от производства.

Вопросы для самоконтроля

1. В чем заключается суть методологии экстремального программирования?

2. Что является неотъемлемой частью разработки при экстремальном программировании?

3. Каким образом происходит тестирование при экстремальном программировании?

4. В чем суть парного программирования и как оно осуществляется?

5. Почему в экстремальном программировании приветствуется простота разработки?

6. В чем преимущество продолжающейся интеграции в разработке программного кода?

7. Что дает участие заказчика в процессе разработки?

8. Что дает при экстремальном программировании применение стандартов кодирования?

РАЗДЕЛ 2 ПРОГРАММИРОВАНИЕ ПРИКЛАДНЫХ ГЕОИНФОРМАЦИОННЫХ ЗАДАЧ

2.1 ЯЗЫК VBA ДЛЯ ПРОГРАММНОЙ ПЛАТФОРМЫ ARCGIS DESKTOP

Реализация технологии COM в ArcObjects

ArcObjects – это платформа разработки для таких модулей ArcGIS, как ArcMap, ArcCatalog и ArcScene. Программные компоненты ArcObjects охватывают полный диапазон функциональных возможностей, доступных в Arclnfo и Arc View для разработчиков программ.

ArcObjects – это инфраструктура, которая позволяет создавать специфичные для данной предметной области компоненты из других компонентов. Компоненты ArcObjects взаимодействуют, чтобы обслужить каждую из функций управления данными и функции представления карты – общие для большинства ГИС приложений. ArcObjects включает более 1200 объектов, которые могут быть использованы для настройки, расширения и построения ГИС приложений.

Структура ArcObjects

ArcObjects построен с использованием технологии **моделей компонентных объектов** фирмы Microsoft (COM – Component Object Model).

Модель компонентных объектов (МКО) – это методология разработки программ. МКО определяет протокол, который соединяет один программный компонент или модуль с другим. Используя этот протокол, можно строить программные компоненты многократного использования, которые могут быть динамически заменяемы в распределенной системе. МКО также определяет модель программирования, известную как интерфейсно-основанное программирование. Компоненты облегчают многократное использование программных компонентов, потому что они – отдельные стандартные блоки, которые могут легко быть собраны в большие системы.

Настройка ArcGIS Desktop

Наиболее общий способ, с помощью которого разработчики настраивают настольные приложения ArcGIS – это использование Visual Basic для прикладных программ, который встроен в ArcCatalog и ArcMap.

С помощью VBA можно усиливать структуру приложений, которая уже существует в этом программном обеспечении, для общего управления данными и задачами, связанных с представлением карт, а также расширять ArcGIS с помощью собственных пользовательских команд, инструментальных средств, меню и модулей.

Используя VBA внутри ArcGIS Desktop, можно удовлетворить большую часть пользовательских потребностей при относительно небольших затратах на разработку. Опытные разработчики в дальнейшем могут расширять ArcGIS Desktop посредством добавления необходимых слоев карты, страниц свойств (атрибутов) и источников данных.

Состав модели VBA

Одна из концептуальных идей Windows и программирования под операционную систему Windows заключается в том, что объекты обмениваются сообщениями. Именно обмен, получение и обработка сообщения являются смыслом существования любого объекта.

ПРИМЕР. Допустим, существует диалоговое окно и кнопка (рис. 2.1). Это окно можно создать, показать методом **Show** и убрать методом **Unload**. Между вызовами этих процедур объект живет, то есть получает и обрабатывает сообщения. Например, при нажатии на кнопку.

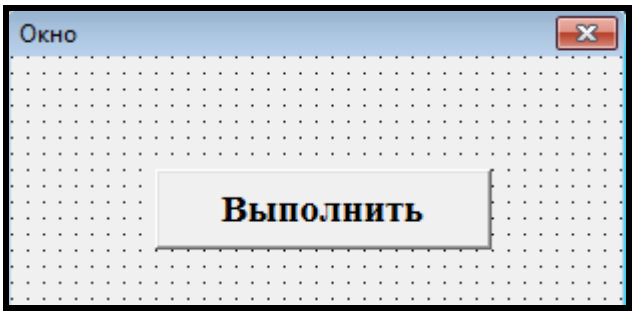


Рисунок 2.1 – Диалоговое окно с кнопкой

Сообщения можно рассматривать, как вызов метода соответствующего объекта. Например, выполняется событие – нажатие на кнопку мышкой. Нажатие состоит из сообщений – мышка двигается, клавиша вниз, клавиша вверх и другие. При этом обрабатывается последовательность сообщений, и система делает вывод о сообщении более высокого уровня – нажата кнопка. В результате вызывается метод. То есть сообщение нажатия на кнопку, вызывает метод **Click** этой кнопки.

Private Sub CommandButton2_Click()

…

End Sub

В общем, сообщения примерно так работают и в реальном мире. Допустим, студент сообщает преподавателю, что он не готов к занятию и у преподавателя будет вызван соответствующий метод. Реализация этого метода зависит от конструкции объекта «преподаватель». Будет какая-то реакция со стороны преподавателя. Для получения реакции нужно послать сообщение.

Модель VBA подразумевает три составляющих (рис. 2.2):

- визуальная;
- системная;
- обработчик событий.

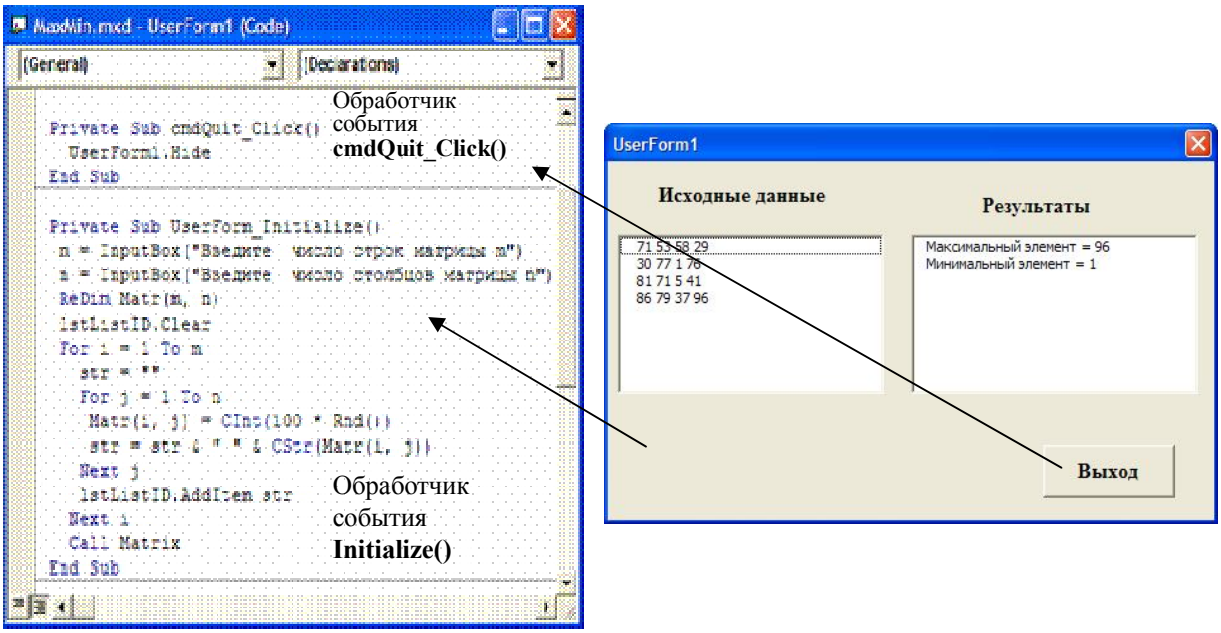


Рисунок 2.2 – Составляющие модели VBA

Визуальная часть – это то, что видно на экране, т.е. интерфейс пользователя. Это окна диалога, кнопки, списки и прочее. При работе с программой пользователь нажимает кнопки, двигает окна и производит различные действия. Он использует объекты интерфейса (элементы управления) для генерации событий.

Системная часть определяет соответствующее событие и формирует сообщение объекту (вызывает метод объекта). Системная часть включает в себя:

- средства операционной системы;
- средства языка программирования.

Обработчик событий – это код, который будет вызван при возникновении события. Код происходит от визуального интерфейса, то есть сначала создается интерфейс, а потом код реализации (листинг).

Вопросы для самоконтроля

1. Поясните структуру ArcObjects. Для чего используются ее компоненты?

2. В чем суть технологии моделей компонентных объектов?

3. Какие существуют способы настройки настольных приложений ArcGIS?

4. В чем заключается смысл существования любого объекта в операционной системе Windows?

5. Какие составляющие подразумевает модель VBA? Охарактеризуйте эти составляющие.

2.2 ТЕХНОЛОГИЯ COM В ARCOBJECTS

Основные понятия технологии COM

В Arclnfo разработчики приложений могут использовать новую COM-технологию разработки ArcObjects. Она предоставляет возможность использования хорошо документированной модели данных с внедряемыми компонентами. ArcObjects позволяет решать широкий круг задач: от настройки и расширения Arclnfo до построения новых ГИС-приложений. Поскольку Visual Basic for Applications – среда разработки, поставляемая с Arclnfo, то он используется наиболее часто.

Технология COM (**Component Object Model** – компонентная модель объектов) предоставляет возможность одной программе (клиенту) работать с объектом другой программы (сервера). COM – это модель объекта, которая предусматривает полную совместимость во взаимодействии между компонентами, написанными разными компаниями и на разных языках.

С точки зрения COM приложение содержит несколько объектов (в частном случае может быть один объект). Каждый объект имеет один или

несколько интерфейсов. В интерфейсе описаны методы объекта, к которым могут получить доступ внешние программы. Если интерфейсов несколько, каждый из них экспонирует некоторое подмножество методов, выполняющих однородные функции.

Объект является частью сервера COM. Сервером может быть исполняемый файл или библиотека DLL. При установке сервера в среде Windows в системный реестр заносится информация обо всех его объектах. Эта информация включает в себя идентификатор класса **CLSID** (**Class Identifier**), однозначно определяющий класс объекта. Заносится информация о типе сервера: внутренний (**in-process** – внутри процесса) – **DLL**, подключаемый к клиенту, локальный (**local**) – работающий отдельным процессом на компьютере клиента, удаленный (**remote**) – работающий на удаленном компьютере. Для внутренних и локальных серверов в реестр заносится полное имя файла, а для удаленных – полный сетевой адрес. Таким образом, в системе хранится вся информация о сервере **COM**, необходимая для вызова его в нужный момент.

Внутренним сервером является DLL (динамически присоединяемая библиотека), которая экспортирует автоматные объекты. Поскольку автоматные объекты поставляются из DLL, а не из других приложений, они являются частью приложения клиента. Это избавляет от больших накладных расходов, сопутствующих каждому вызову автоматного сервера.

Локальный или удаленный сервер – это автономный исполняемый файл, экспортирующий автоматные объекты. Примером этого мог бы быть ArcMap. ArcMap имеет объекты, которые он экспортирует в качестве автоматных.

Внешние приложения, обращающиеся к объекту COM, являются клиентами COM. Клиент получает указатель на интересующий его интерфейс объекта и через этот указатель может вызывать методы объекта. Спецификация COM запрещает изменять однажды объявленный интерфейс. Это обеспечивает нормальную работу клиента при любых модификациях сервера.

Таким образом, клиенту достаточно знать интерфейсы объекта и предоставляемые ими методы, а о остальном позаботится система. В нужный момент она запустит сервер COM, если он еще не был запущен, сервер создаст объект, объект загрузит все необходимые ему данные и клиенту вернутся указатели на объект и его интерфейсы, с которыми он может

работать. Система позаботится также о том, чтобы обеспечить работу объекта сразу с несколькими клиентами. Для этого она ведет учет числа ссылок на объект. При выдаче клиенту указателя на интерфейс число ссылок увеличивается на единицу. А при окончании работы клиента с объектом число ссылок на единицу уменьшается. Если число ссылок стало равно нулю, система уничтожает объект, с которым в данное время не работает ни один клиент.

Вся информация об объектах COM, их интерфейсах, свойствах, методах, параметрах методов содержится в библиотеке типов, которая создается разработчиком объекта COM и распространяется вместе с объектом. Библиотека создается с помощью языка описания интерфейса IDL (**Interface Definition Language**).

Интерфейсы

Каждый интерфейс имеет имя, начинающееся с символа «I», и **GUID** – глобальный уникальный идентификатор (**Globally Unique Identifier**). Подобные **GUID** создаются и используются не только для интерфейсов. Для интерфейсов **GUID** называется **IID.**

Каждый объект **COM** имеет интерфейс **IUnknown.** Этот интерфейс имеет всего три метода: **QueryInterface** – получение указателя на интерфейс, **AddRef** и **Release** – увеличение и уменьшение на 1 числа ссылок на объект.

Метод QueryInterface возвращает указатель на интерфейс с заданным IID. Метод Release должен вызываться по окончании работы с интерфейсом, чтобы уведомить объект, что данный клиент в нем более не нуждается. Эти два метода используются всегда при работе с объектом COM.

Метод AddRef используется только в тех случаях, когда один клиент передал другому ссылку на интерфейс. Поскольку при этом метод AddRef, автоматически увеличивающий число ссылок, не вызывается, клиент, которому передана ссылка, должен вызвать AddRef, чтобы доложить объекту, что он тоже с ним работает.

Все остальные интерфейсы объектов являются наследниками IUnknown и наследуют те же три метода, добавляя, конечно, свои собственные. Среди интерфейсов-наследников необходимо отметить **IDispatch**. Это интерфейс, используемый клиентами для доступа к методам и свойствам объекта.

Библиотека COM

Технология COM реализуется специальными библиотеками. Они содержат стандартные интерфейсы и функции, обеспечивающие создание объектов COM и управление ими.

Имена всех функций библиотеки начинаются с «**Co**». Создание объекта осуществляется вызовом функции CoCreateInstance этой библиотеки и передачей в нее CLSID требуемого класса, IID интерфейса и требуемого типа сервера. Библиотека COM обращается к системному реестру, в котором хранится информация о сервере, запрашивает его и возвращает требуемый указатель.

Так как технология COM не зависит от языка, в ней используются типы, отличные от других языков. Прежде всего, это относится к строкам, которые в разных языках описываются по-разному. В COM используется свой строковый тип – BSTR (Basic String). Он описывает строку, в начале которой указана ее длина. Поскольку длина строки известна, завершающего нулевого символа не требуется.

Реализация технологии COM в ArcObjects

ArcObjects – это платформа разработки для таких модулей ArcGIS, как ArcMap, ArcCatalog и ArcScene. Программные компоненты ArcObjects охватывают полный диапазон функциональных возможностей, доступных в Arclnfo и Arc View для разработчиков программ.

ArcObjects – это инфраструктура, которая позволяет создавать специфичные для данной предметной области компоненты из других компонент. Компоненты ArcObjects взаимодействуют, чтобы обслужить каждую из функций управления данными и функции представления карты, общие для большинства ГИС приложений. ArcObjects включает более 1200 объектов, которые могут быть использованы для настройки, расширения и построения ГИС приложений.

Модель компонентных объектов

ArcObjects построен с использованием технологии «модели компонентных объектов» фирмы Microsoft (COM).

Модель компонентных объектов (МКО) – это методология разработки программ. МКО определяет протокол, который соединяет один программный компонент или модуль с другим. Используя этот протокол,

можно строить программные компоненты многократного использования, которые могут быть динамически заменяемы в распределенной системе. МКО также определяет модель программирования, известную как интерфейсно-основанное программирование. Компоненты облегчают многократное использование программных компонентов, так как они состоят из отдельных стандартных блоков, которые могут легко быть собраны в большие системы. Поэтому можно расширять состав ArcObjects путем написания COM-компонент, используя любой COM-совместимый язык разработки.

Чтобы понять МКО, важно понять, что это не объектно-ориентированный язык, протокол или стандарт. Модель компонентных объектов – больше чем просто технология, это – методология разработки программ. МКО определяет протокол, который соединяет один программный компонент или модуль с другим. Используя этот протокол, можно строить программные компоненты многократного использования, которые могут быть динамически заменяемы в распределенной системе. МКО также определяет модель программирования, известную как интерфейсно-основанное программирование. Компоненты облегчают многократное использование программ, потому что они – отдельные стандартные блоки, которые могут легко быть собраны в большие системы.

МКО определяет объектную модель и требования программирования, которые дают возможность объектам COM взаимодействовать с другими объектами COM. Эти объекты могут быть в пределах одного процесса, в различных процессах, или даже на удаленных компьютерах. Они могут быть написаны на разных языках и, возможно, разработаны очень разными способами.

Вопросы для самоконтроля

1. Поясните суть технологии COM.
2. В чем суть методологии компонентных объектов?
3. Что собой представляет любое приложение с точки зрения технологии COM?
4. Что собой представляет DLL библиотека и для чего она используется?
5. Что такое интерфейс объекта и каково его назначение?

2.3 МОДУЛИ, ОБЪЕКТЫ И КЛАССЫ

Ранее рассматриваемые примеры программ выглядели весьма тривиально. Весь код, с которым приходилось работать, содержался в процедурах событий форм. Но для того, чтобы пользоваться расширенными возможностями Visual Basic, необходимо отказаться от этого ограниченного подхода. Для этого необходимо создавать собственные процедуры и модули.

Процедуры

Процедура – это совокупность операторов, выполняющих определенные действия. Процедуры имеют стандартное оформление:

Public {[**Private**], [**Static**]} **Sub** Name (Список аргументов)
тело процедуры
End Sub

Public – глобальная процедура, доступна для всех других процедур во всех модулях проекта.

Private – процедура модуля, доступна для всех других процедур в данном модуле.

Static – служебное слово, которое говорит о том, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры.

Name – имя процедуры, удовлетворяющее стандартным правилам написания имен в VBA. Имя процедуры обработки события состоит из имени объекта и имени события.

Список аргументов – список формальных параметров (аргументов) процедуры **Sub** (имен переменных, которые должны быть переданы в процедуру при обращении к ней).

Синтаксис элемента структуры **Список аргументов**:

[**Optional**] [**ByVal**, **ByRef**] [**ParamArray**] Имя переменной **As** Тип

Optional – ключевое слово, указывающее, что аргумент не является обязательным. Все аргументы, описанные как **Optional**, должны быть типа **Variant**.

ByVal – указывает на то, что этот аргумент передается по значению.

ByRef – указывает на то, что этот аргумент передается по ссылке на его адрес в памяти. Описание **ByRef** используется по умолчанию.

ParamArray – используется только в качестве последнего элемента в списке аргументов для указания о том, что конечным аргументом в списке параметров процедуры является массив значений, описанный как тип **Variant,** т. е. это позволяет задавать произвольное количество аргументов в процедуре.

Для лучшего понимания того, как написать процедуру и использовать ее в программе, создадим процедуру, заменяющую стандартный указатель-стрелку манипулятора мышь песочными часами. Ее следует вызывать перед тем, как какая-либо программа займется выполнением какой-нибудь длительной задачи. Указатель в виде песочных часов подскажет пользователю, что программа не «зависла», а производит расчет.

ПРИМЕР. Процедура создания указателя мыши в форме песочных часов

```
Public Sub ShowHourdlass()
  Screen.MousePointer = 11
End Sub
```

Процедура создает указатель мыши в виде песочных часов. Однако после того, как указатель был изменен, мы должны иметь возможность вернуть ему прежний вид.

```
Public Sub ShowMousePointer()
  Screen.MousePointer = 0
End Sub
```

Процедуры ShowHourglass и ShowMousePointer необходимы в тех случаях, когда программа занята, а пользователю приходится ждать ее освобождения. Вызовите ShowHourglass в начале, а ShowMousePointer – в конце периода ожидания.

ПРИМЕР. Процедура вычисления кубического корня

```
Public Sub CubeRoot (x As Double, y As Double)
  If x = 0 Then
    y = 0
```

```
    Exit Function
  End If
    y = 10 ^ ((Log (Abs (x)) / Log (10)) / 3)
    If x < 0 Then
      y = - y
    End If
End Sub
```

Вызов процедуры

```
A = 16
CubeRoot A, Z
```

Переменная Z получит значение 2.

Функции

Функции во многом похожи на процедуры. Существует лишь одно принципиальное отличие – при вызове они возвращают значение. Функция получает один или несколько объектов данных, называемых аргументами, и выполняет с ними некорые действия. Их результат возвращается функцией. Функции имеют вид:

```
Function Имя_Функции [(Список аргументов)] As Тип
  тело функции
End Function
```

Лучше всего рассмотреть работу функций на конкретном примере. Допустим создадим функцию вычисления кубического корня.

ПРИМЕР. Функция вычисления кубического корня:

```
Public Function CubeRoot (x As Double) As Double
  If x = 0 Then
    CubeRoot = 0
Exit Function End If
    CubeRoot = 10 ^ ((Log (Abs (x)) / Log (10)) / 3)
    If x < 0 Then
      CubeRoot = - CubeRoot
    End If
End Function
```

Вызов функции

A = 16

Z = CubeRoot (A)

Переменная Z получит значение 2.

Рассмотрим приведенный выше код. Функция получает аргумент **x** и вычисляет его кубический корень. Тип значения, возвращаемого функцией (в примере **Double** – числовой тип, который может работать с очень большими и очень малыми, а также дробными числами), указывается в заголовке функции после ключевого слова **As**.

Текст функции заканчивается командой **End Function**, а не **End Sub**. Обратите внимание на три строки, в которых присваивается значение переменной **CubeRoot**. Значение функции возвращается в виде переменной, имя которой совпадает с именем функции (в примере – **CubeRoot**). Если бы функция называлась, допустим, **TimesTwo** (), то и значение возвращалось бы в переменной **TimesTwo** (например, TimesTwo = x × 2).

Перед тем как приступать к обработке аргумента, желательно проверить полученные данные. Сначала проверим, равен ли аргумент 0 – как известно, кубический корень из 0 равен 0, поэтому можно сразу присвоить переменной **CubeRoot** возвращаемое значение и выйти из функции, не выполняя дальнейших вычислений. Для выхода из функции применяется команда **Exit Function**.

Команда **CubeRoot** = 10 ^ ((Log (Abs (x)) / Log (10)) / 3) возвращает кубический корень аргумента **X**.

Теперь проверим, не является ли аргумент функции отрицательным числом. Приведенная выше формула всегда возвращает положительный результат, поэтому для отрицательного аргумента необходимо изменить знак возвращаемого значения на противоположный. В Visual Basic это делается командой:

CubeRoot = - CubeRoot
Then ... действия вашй программы ... **End If**

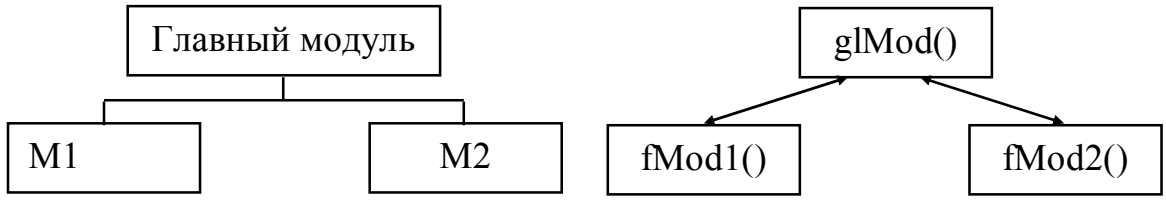
Модули и классы

Принципы модульного программирования

В основу модульного программирования положен принцип функци-

ональной декомпозиции. Сложная задача разбивается на подзадачи, а программа на модули (рис. 2.3, а). На верхних уровнях находятся управляющие модули. Они организуют вызовы модулей нижних уровней. На нижних уровнях находятся исполнительные модули (например, M1 и M2). Каждый из них решает определенную подзадачу.

В соответствие с иерархической схемой программы разрабатываются основной и вспомогательные алгоритмы модулей. Основной алгоритм реализует функции главного модуля, а вспомогательные алгоритмы – модули нижних уровней.



а б

Рисунок 2.3 – Модульная структура алгоритма (а) и программы (б)

Между модулями организуется связь по управлению и информации.

Связи по управлению осуществляются по вертикали сверху вниз. Модули верхних уровней вызывают модули нижних уровней. Передача управления вызываемому модулю осуществляется через его начало, а выход из вызываемого модуля происходит через его окончание. Вызываемый модуль возвращает управление вызвавшему его модулю. Модули нижних уровней не могут вызывать модули верхних уровней. Модули одного уровня не могут вызывать друг друга.

Связи по информации осуществляются либо через глобальные переменные, которые доступны во всех модулях, либо путем передачи параметров. Объявленные в модуле переменные являются локальными переменными. Они не доступны другим модулям.

Модули VBA реализуются процедурами и функциями. Изобретение функций – это большой шаг в эволюции языков программирования. Функции являются строительными блоками, из которых строят сложные программы. В основу появления функций положен принцип обозначения сложных комплексных сущностей простыми именами, который присущ

человеческому мышлению. В процессе познания люди систематизируют свои знания, объединяют их в логические конструкции и дают им названия. Из этих конструкций собираются еще более сложные конструкции, которым тоже дают названия. Аналогично обстоит дело и с функциями.

Если формы и элементы два важнейших «строительных блока» Visual Basic, то третьим видом «блоков» являются программные модули (или просто модули). Программный модуль представляет собой текстовый ASCII-файл, содержащий процедуры, функции, переменные или константы. Модули представляют собой текстовые ASCII-файлы с программным кодом. В них удобно группировать взаимосвязанные процедуры, которые могут использоваться в программе.

Код проекта может состоять из множества программных модулей.

Классы представляют собой основные строительные блоки объектно-ориентированного программирования – модели, в которой программа описывается в виде совокупности объектов. Класс не только позволяет выделить часть функциональных средств программы в отдельный объект, но и расширяет возможности базовых модулей – можно защитить одни фрагменты программы, а другие предоставить в распоряжение программы. Этот процесс называется **инкапсуляция.**

Программные объекты, конструируемые в ООП, имитируют поведение объектов реального мира. Хорошо спроектированный класс представляет собой вполне самостоятельный фрагмент программы. Это означает, что вы можете перенести класс из одного проекта в другой, и он будет нормально работать без каких-либо исправлений.

Поскольку модули содержатся в отдельных файлах, их можно включать сразу в несколько проектов. Таким образом, появляется возможность повторно использовать написанный код. Например, существует несколько модулей, содержащих взаимосвязанные процедуры. В одном модуле хранятся функции для работы с готовыми окнами. В другом модуле объединены функции, упрощающие работу с мультимедиа-устройствами. Группируя однородные функции в пределах одного модуля, можно создать программную библиотеку. Если поместить модули в один каталог или логически связанный набор подкаталогов, их можно будет использовать в других проектах.

Использование программных библиотек экономит время, поскольку нет необходимости заново создавать уже написанный код. Различные типы

данных находятся в одной библиотеке, но при этом не смешиваются друг с другом. Другое преимущество программных библиотек заключается в том, что после отладки их содержимое можно использовать в других проектах. В итоге создается набор инструментов, который можно использовать при работе над разными проектами.

Создание модулей

Модуль – это файл с текстом программы, вставленный в документ. Программа на языке VBA состоит из одного или нескольких модулей.

Модули могут быть следующих типов:

- модули, связанные с объектами приложения, реагирующие на события (модули-обработчики событий);
- программные (стандартные) модули классов, создаваемые программистом;
- модули макросов, создаваемые **Macrorecorder.**
- модули-обработчики событий всегда связаны с объектами, которые реагируют на события.

В модули следует помещать только те процедуры и функции, которые непосредственно обрабатывают события. Программные (стандартные) модули – основной вид модулей, который создается программистом. Большую часть всех процедур и функций следует размещать в стандартные модули.

Практика показывает, что целесообразнее создавать не один большой модуль, а несколько маленьких. В один модуль следует помещать набор функций, совместно вызывающих друг друга, объединенных общей темой и общей функциональной направленностью.

Реально всегда выделяется главный модуль, в котором сосредоточено описание глобальных переменных.

Модуль класса – специальный тип модуля, который используется при создании классов, разрабатываемых программистом для решения своих задач. В VBA предусмотрена возможность создания пользовательских объектов, которые являются экземплярами классов. Все объекты одного и того же класса используют одинаковые методы в ответ на одинаковые сообщения. Классы конструируются в модулях классов, которые создаются командой **Insert**/**Class Module** и записываются в появившемся окне. Нажав клавишу **F4**, можно выбрать имя класса.

Имя модуля класса является именем класса объектов. В разделе описания модуля объявляются переменные уровня модуля, которые используются как «значения свойств».

Класс инициализируется при помощи процедуры **Class_Initialaize.** Имена свойств, значениями которых являются числовые данные, объявляются при помощи процедуры **Property Let.** Процедурой **Property Set** объявляются имена свойств, значениями которых являются объекты.

При помощи процедуры **Property Get** устанавливается возможность считывания значений свойств. Методы создаются при помощи обычных процедур и функций.

Вопросы для самоконтроля

1. Что такое процедура и каков формат оформление процедур в VBA?
2. В каких целях используются функции их основное отличие от процедуры?
3. Какой проинцип положен в основу модульного программирования и в чем он заключается?
4. Каким образом реализуются модули в VBA?
5. Поясните суть процесса инкапсуляции.
6. Каких типов могут быть модули? Охарактеризуте эти типы.

2.4 НЕКОТОРЫЕ РЕСУРСЫ ДЛЯ РАЗРАБОТЧИКОВ

Получение справки об объектной модели важно для успешной работы с ArcObjects. Диаграммы классов наиболее полезны при проектировании на начальной стадии разработки прикладных модулей. Это позволяет разработчикам оценить полную структуру объектной модели, предоставляемой через ArcObjects. Полезно ознакомиться и с PDF файлами, включенными в поставку программного обеспечения.

Программы просмотра объектов

В дополнение к диаграммам классов, содержащихся в PDF файлах, информация о библиотеках типов может быть просмотрена, с использованием ряда программ просмотра объектов. Visual Basic имеет встроенную программу просмотра объектов. Программа OLEView (утилита от Microsoft) также отображает информацию о библиотеках типов. Для просмотра объектов из библиотеки ESRI Object Library рекомендуется про-

грамма ESRI Object Viewer. Информация относительно классов и интерфейсов может быть отображена в формате Visual Basic, Visual C++, или в виде объектной диаграммы.

Программы просмотра объектов могут использоваться для рассмотрения классов, но не могут использоваться, чтобы рассмотреть абстрактные классы. Абстрактные классы можно также просматривать на объектных диаграммах, где их использование должно упростить модели.

Справка по компонентам

Все интерфейсы и абстрактные классы подробно документированы в справочном файле по компонентам. Это скомпилированный файл HTML, который можно посмотреть отдельно или при использовании интегрированной среды разработчика.

Дополнительную информацию можно почерпнуть из входящей в комплект ArcGIS книги ESRI «Изучение ArcObjects» (Exploring Arcobjects), которая рассматривает каждую из этих подсистем более детально. Каждая глава в книге посвящена отдельной подсистеме и включает обсуждение по каждому объекту и интерфейсу подсистемы с многочисленными примерами кода, показывающего их использование. Диаграмма модели объектов ArcGIS иллюстрирует упрощенные объектные модели для каждой подсистемы в ArcObjects и является хорошей отправной точкой для знакомства с каждой из подсистем.

Рассмотрим краткую информацию о подсистемах ArcObjects. Каждая подсистема имеет описание своей объектной модели в виде файла PDF.

3D Analyst

Компоненты в дополнительном модуле 3D Analyst обеспечивают структуру для трехмерной визуализации и моделирования поверхности.

Application Framework

Объекты в этом модуле позволяют программно настраивать интерфейс пользователя в ArcMap и ArcCatalog.

ArcCatalog

Расширяемая архитектура ArcCatalog дает возможность разработчикам расширить объектную модель, чтобы добавлять пользовательские объекты и пользовательские представления в приложении.

ArcMap

Эти объекты обеспечивают основные функциональные возможности для приложения ArcMap и используются для отображения и управления документами карты.

ArcMap Editor

Подсистема редактора ArcMap содержит объекты, облегчающие редактирование покрытий, шейп-файлов и баз геоданных.

Display

Объекты Display выполняют основные функции для показа символики карты, отображая графическое редактирование объектов карты, выполняя преобразования координат и управляя экраном дисплея.

Geocoding

Объекты Геокодирования обеспечивают функциональные возможности для геокодирования адресов, также как для рассмотрения и поддержания результатов геокодирования. Эти объекты также обеспечивают функциональные возможности для создания, управления, и поддержки служб геокодирования.

Geodatabase

Объекты Базы Геоданных обеспечивают способность создавать и управлять данными в базах геоданных. Эти объекты также обеспечивают механизмы для доступа и выполнения запросов к географическим данным, сохраненным в базах геоданных, шейп-файлах и покрытиях.

Geometry

Geometry – подсистема, которая обрабатывает геометрию, или форму векторных данных, сохраненных в классах векторных данных или других графических элементах.

IMS

Объекты IMS обеспечивают функциональные возможности для соединения с серверами ArcIMS и получения доступа к службам изображений или векторных данных ArcIMS. Также, через эти объекты разработчик может обращаться к слоям изображений/векторных данных и их свойствам. Примером такого свойства может быть символика уровня.

Network

Объекты Network обеспечивают функциональные возможности для создания, управления и операций анализа с использованием пространственных сетей. Имеются также объекты для создания пользовательских решений и настройки сервисного расширения **Сетевого анализа**.

Output

Печать и экспорт файла выполняется через объекты в этой подсистеме вывода.

Raster

Объекты Raster обеспечивают доступ к управлению растровыми данными на диске и в памяти. Эти объекты также обеспечивают средства для классификации растровых данных и управления визуализацией этих данных.

Spatial Reference

Подсистема Spatial Reference – набор компонентов, которые обеспечивают интерфейсы к различным функциональным возможностям пространственных ссылок.

Вопросы для самоконтроля

1. Для чего используются диаграммы классов при проектировании прикладных модулей?
2. В каких целях используются встроенные в Visual Basic программы просмотра объектов?
3. Какие подсистемы ArcObjects вы знаете? Охарактеризуйте каждую из них.
4. Что иллюстрирует диаграмма модели объектов ArcGIS?
5. На каких диаграммах можно просматривать абстрактные классы?
6. В каких целях они используются?

2.5 КОМПОНЕНТ КАРТА КАК ОБЪЕКТ ГЕОИНФОРМАЦИОННОЙ СИСТЕМЫ

С помощью карт местности решаются такие задачи, как наложение на тематические данные линий высот, информации об уклонах местности и экспозиции (освещенности). Разнородность обрабатываемых данных огромная.

Геометрические данные. Геометрия и топология

Геометрия – раздел математики, в котором изучаются пространственные отношения (взаимное расположение), формы (геометрические тела) и их обобщения.

К геометрическим свойствам относятся: длина, площадь, объем, форма и другие (регулярность, ориентировка, наличие центра, уклон).

Топология – раздел математики, изучающий топологические свойства фигур, т.е. свойства, не изменяющиеся при любых взаимно-однозначных и непрерывных отображениях.

К топологическим свойствам фигур относится: размерность, ориентация, инцидентность, смежность, связность. Такие фигуры, как окружность, эллипс, контур квадрата топологически идентичны.

Геометрия пространственных объектов полностью описывается через форму и относительное положение тел. Для этих описаний можно воспользоваться расстояниями и углами, но обычно пользуются координатами, тип которых определяет системы отношений и метрик.

Внутри топологии важен только факт, что точки и линии находятся в определенных взаимных отношениях, а не геометрическая форма этих объектов. Точка (топологический узел) – это носитель графической информации. Линии и плоскости могут рассматриваться как следствие объединения характерных точек в группы. Форму связующих элементов можно определить через дополнительные предписания, например, дугу через радиус. Носителем топологической информации является граница.

Главная разница между геометрией и топологией состоит в инвариантности топологического построения к топологическим трансформациям, изменяющим геометрические построения.

Геометрические характеристики – это метрические характеристики, то есть те, которые можно измерить, для чего необходимо выбрать единицу и способ измерения. Выражаются измерения обычно в численных значениях, для них характерны отношения порядка – больше, меньше. Результаты измерения в общем случае зависят от принятой метрики (положение начала координат, выбор единицы измерения). Они могут меняться при любых изменениях формы объекта, изменении положения начала координат.

Топологические свойства не являются метрическими и поэтому не зависят от выбора системы отсчета координат, изменений формы, которые описываются как взаимно-однозначные и непрерывные отображения.

Простейшим примером разницы между геометрией и топологией (в ГИС) может служить план системы транспорта. Изображение линий связи между транспортными узлами на схематическом плане – топологического характера, так как речь идет только о наличии соединений.

И геометрия, и топология основываются на базовых геометрических элементах, перечисленных выше. Геометрия объекта в ГИС может быть представлена посредством точек, линий и плоскостей.

Вектор и растр

Описание с помощью точек и линий приводит к понятию векторных данных, в то время как регулярное плоскостное описание ведет к растровым данным. Например, поле изображается векторным способом через точки границ, а сбор данных с помощью спутника приводит к растровому способу представления.

При векторном способе представления данных топология должна быть представлена эксплицинтно (во всех параметрах), при растровом способе представления данных она приводится в порядок через строки и столбцы, ее можно представить как структуру дерева.

Растровый и векторный способы представления данных существуют в пространственных информационных системах рядом друг с другом. В каждом конкретном способе используются его преимущества, например, растровые данные, существенно лучше подходят к описанию плоскостных явлений, в то время как векторные – сильнее в линейных построениях.

Под векторными данными понимают описание пространственных объектов в виде набора координат и их взаимосвязей. Их основными элементами являются точка, линия и область. Векторные данные могут быть векторно-топологическими или не топологическими данными.

Векторные данные имеют значение на всей масштабной шкале ГИС, они доминируют в области крупных масштабов от 1:100 до 1:10000. Основные области их применения – кадастр недвижимости, учет земельных угодий, документация по линиям электропередач и планирование, при котором сбор данных происходит путем геодезической съемки и вычислений, перевода на язык ЭВМ аналоговой графической информации.

Для **векторных данных** существенны следующие свойства:

- точка, линия основные графические структуры, область – рассматриваются как плоскость, ограниченная совокупностью замкнутых ли-

ний; объем – как пространство, ограниченное областями;

- логическое структурирование данных и отношения к объекту легко осуществимы;
- сбор данных производится по точкам, но время сбора происходит с высокой скоростью;
- малый массив данных, короткое время вычислений;
- векторные данные можно представить как геометрический граф.

Граф однозначно определяется через точки (узлы) и множество линий (дуг), а также через конкретное отображение дуг на узлы. Внешняя геометрия, то есть положение узлов и форма дуг играют роль только для геометрической версии. Графы в целом определяются независимо от метрики. Характерным для графа является исключительно его топология, в форме структурных отношений между узлами и дугами.

На вопрос от каких узлов отходят по меньшей мере три дуги, можно ответить только тогда, когда полностью понятны топологические отношения. Применение топологии способствует расширению области применения ГИС.

Растровыми данными называется цифровое представление пространственных объектов в виде совокупности значений, полученных в узлах регулярной сети. В противоположность векторным данным растровое изображение относится непосредственно к плоскостям, а не к линиям. Это изображение пространственного объекта – самая первая форма геометрического изображения. Основной геометрический элемент – пиксель (Picture Element – элемент картинки), который располагается в виде квадратных или прямоугольных элементов одной формы в матрице. Эти элементы однородно заполняют область.

Растровые данные не различают точку, линию или плоскость, то есть между отдельными элементами изображения не существует логических связей. Растровые данные оцениваются исключительно по свойствам пикселя (серый или цветной, высота, эмиссия и т.д.).

Основные способы растровой обработки данных находится в среднем диапазоне масштабов (от 1:10000 до 1:1000000), при этом сбор данных происходит путем сканирования земной поверхности с помощью специальных камер, установленных на спутниках, или с помощью других устройств (изображения с воздуха, ортофотографии). В исключительных случаях растровые данные используются и для крупных масштабов,

например при оценке земель и в сельском хозяйстве, чтобы определить качество земель и урожайность. Растровые данные характеризуются следующими свойствами:

- пиксель – основная графическая структура;
- возможность обзора характеристик всей плоскости;
- порядок определяется позицией пикселя;
- логическое структурирование и отношение к объекту ограничены;
- простота сбора данных, короткое время сбора;
- большие массивы данных – высокая стоимость вычислений.

Графические данные

Графические данные это – это совокупность геометрических и графических атрибутивных (описательных) данных, необходимых для представления определенного геометрического объекта на графическом устройстве вывода (принтер, плоттер, монитор). Графические данные могут быть представлены в аналоговой форме (карта или подобное карте изображение) и в цифровой форме (компьютерная графика).

Графические **атрибутивные данные** (графические атрибуты) – это данные о способах представления пространственного объекта определенной тематики на графическом устройстве вывода. К графическим атрибутам относятся:

- цвет;
- форма представления точечных объектов (вид знака, символа);
- вид линий (непрерывная, пунктир);
- толщина линии (1 пиксель, 0,25 мм);
- стиль области (стиль границы, стиль заливки);
- наличие, стиль и положение текста.

Вопросы для самоконтроля

1. Какие задачи решаются с помощью карт местности.
2. В чем разница между геометрическими и топологическими характеристиками объектов? Приведите примеры.
3. В чем отличия растрового и векторного способа представления данных?
4. Для решения какого рода задач чаще используются векторые данные?

5. Какими свойствами обладают векторные данные?

6. Что можно сказать о основном геометрическом элементе растрового изображения?

7. Какимим свойствами характеризуются растровые данные?

8. Охарактеризуйте графические атрибутивные данные? Что к ним относится? Приведите примеры.

2.6 ОРГАНИЗАЦИЯ ДОСТУПА К КАРТАМ И ТЕМАТИЧЕСКИМ СЛОЯМ. АЛГОРИТМЫ РАЗМЕЩЕНИЯ СЛОЕВ В КАРТЕ

В современных информационных системах можно выделить два основных типа данных: пространственные и атрибутивные (семантические). Например, в системе корпоративного учета недвижимости пространственные данные об объектах недвижимости хранятся в тематических слоях геоинформационной системы, а атрибутивные данные, описывающие права, субъекты прав, правоустанавливающие и правоудостоверяющие документы – в системе управления базами данных (СУБД). Поскольку один объект может описываться как атрибутивными, так и пространственными данными, разработчики информационных систем сталкиваются с необходимостью осуществлять связь пространственных и атрибутивных данных.

С появлением ArcGIS и базирующейся на COM-технологии платформы ArcObjects разработчики получили превосходный набор инструментов для разработки ГИС решений.

Для интеграции пространственной и атрибутивной информации ArcGIS предоставляет разработчикам ряд возможностей:

1. Разрабатывать информационную систему для атрибутивных данных (АИС) непосредственно в ArcGIS с помощью встроенной среды Visual Basic for Applications (VBA) (рис. 2.4).

2. Разрабатывать АИС с помощью стандартных средств разработки (Delphi, C++, VB) и внедрить в приложение объект MapControl как ActiveX объект. При этом доступ к функциям ГИС будет обеспечен с помощью интерфейсов объекта MapControl.

3. Разрабатывать АИС независимо от ГИС-приложения с возможностью их взаимодействия.

4. Первые два подхода имеют ряд недостатков:

5. ограниченность VBA по сравнению со специализированными средствами разработки;

6. неудобство использования MapControl по сравнению со связкой ArcMap и VBA;

7. избыточность созданной системы, если пользователь работает только с одним типом данных.

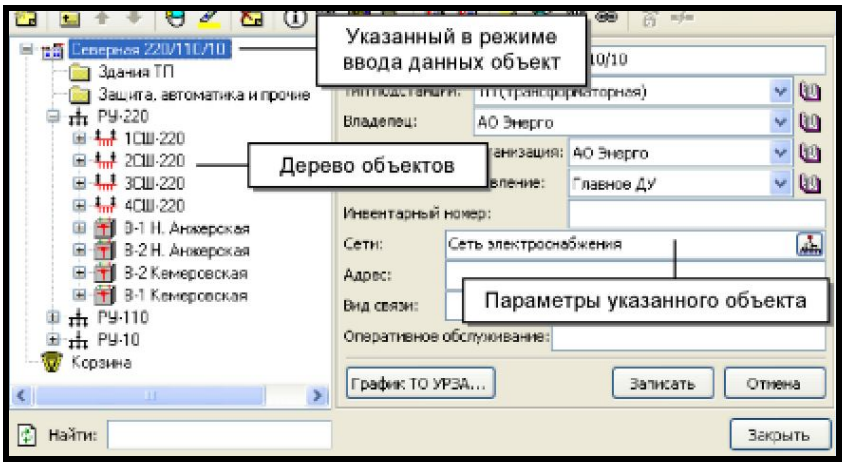


Рисунок 2.4 – Пример интерфейса АИС, разработанной в ArcGIS

Использование первого варианта оправдано, когда в разрабатываемой системе основное внимание уделяется работе с пространственными данными, а к вводу либо просмотру атрибутивных данных не предъявляются высокие требования.

Второй вариант имеет смысл использовать, если в разрабатываемой системе основное внимание уделяется работе с атрибутивными данными, а задачи, связанные с пространственной информацией, относительно простые (отображение объектов на карте, поиск).

Третий вариант более универсален. Разработчику предоставляется возможность создавать приложения независимо, максимально эффективно использовать ArcGIS и средства разработки информационных систем, минимально изменять код существующих информационных систем и ГИС приложений для обеспечения их взаимодействия. Пользователи такого программного комплекса могут вводить информацию независимо, на различных рабочих местах, эффективно используя лицензии ArcGIS.

ПРИМЕР. Создание и поддержание взаимно однозначного соответствия между записями в таблице атрибутивной базы данных и объектами

пространственной базы данных. Например, записи из таблицы «Земельный участок» базы данных объектов недвижимости может быть поставлен в соответствие пространственный объект из слоя «Земельные участки» и наоборот. В рамках рассматриваемой задачи создаваемая АИС должна решать следующие функции:

- Запрос по атрибутивным данным – пользователь выбирает нужные ему земельные участки в АИС, а ГИС отображает эти участки на карте.
- Запрос по пространственным данным – пользователь выбирает нужные ему участки в ГИС, а АИС предъявляет эти участки.
- Привязка данных – пользователь устанавливает соответствие между участками в пространственной и атрибутивной базах данных.

Связь между пространственной и атрибутивной информацией

Первоначально разработчик должен выбрать общий атрибут – поле (ключ), по которому осуществляется связь между пространственной и атрибутивной информацией. В качестве общего атрибута может выступать как «естественный» ключ, например кадастровый номер, так и «искусственный», например поле с автонумерацией (FID). В приводимых ниже примерах в качестве поля связи выбран идентификатор объекта в атрибутивной базе данных.

Также необходимо решить задачу обмена информацией между приложениями. Во времена Arc View 2.x и 3.x единственным способом обмена данными с ГИС было использование механизма динамического обмена данными (DDE). ArcGIS построен на основе технологии Component Object Model (COM), и такие приложения как ArcMap, ArcCatalog и ArcScene являются внешними COM-серверами. Для обеспечения двухстороннего взаимодействия приложений необходимо, чтобы АИС тоже являлась COM-сервером.

Предположим, что данные в атрибутивной и пространственной базах сформированы должным образом (связь между пространственной и атрибутивной подсистемами комплекса уже установлена), и реализуем задачи запроса по атрибутивным и пространственным данным.

ПРИМЕР. ГИС должна отобразить на карте объект, выбранный в АИС.

Для этого в программе ArcMap необходимо создать VBA процедуру, которая выделяет объект в заданном слое по его номеру.

Public Sub SelectMap Feature (Number As String)

Реализация этой процедуры заключается в поиске и выделении объекта с заданным номером.

```
Set pFeatureSelection = pFeatureLayer 'QI
Set pQueryFilter = New QueryFilter
pQueryFilter.WhereClause = "ParcellD = " + Number
pActiveView.PartialRefresh esriViewGeoSelection, Nothing, Nothing
pFeatureSelection.SelectFeatures pQueryFilter, esriSelectionResultNew, False
pActiveView.PartialRefresh esriViewGeoSelection, Nothing, Nothing
AppActivate Application.Caption      'Активация ArcMap
```

Для вызова этой процедуры АИС должен получить интерфейс IVbaApplication, который реализуется объектом Application. Для того, чтобы получить интерфейс IApplication в АИС необходимо создать объект AppROT. IAppROT, реализуемый этим объектом, позволяет получить доступ к IApplication всех запущенных приложений ArcGIS (ArcMap, ArcCatalog, ArcScene). Получить IApplication программы ArcMap с разработанным проектом можно по заголовку (IApplication.Document.Title). Имея нужный IApplication, получим интерфейс к проекту VBA.

```
VbaApp: = ArcApp as IVbaApplication; //QI
```

Передача параметров в функцию VBA осуществляется посредством вариантных массивов.

```
Arg := VarArrayCreate( [0,0], varVariant);
Arg[0] := Number;
VBAApp.RunVBAMacro('Project', 'ThisDocument', 'SelectMapFeatures', Arg, Res);
```

ПРИМЕР. ГИС передает выделенный объект в АИС.

Для этого АИС должна выступать в роли сервера автоматизации (Automation server). Для создания минимальной функциональности достаточно создать всего лишь один COM-класс (CoClass) с одним интерфейсом, который реализует одну функции: запрос к карте QueryByMap.

```
procedure TMyComServ.QueryByMap (Num : OleVariant); begin
MainData.qryObject.Active := False;
MainData.qryObject.SQL.Text:= 'SELECT*From Object WHERE idObject= '+NumStr;
MainData.qryObject.Active := True;
Application.BringToFront;  'Активация приложения
end;
```

Для вызова этой функции в ArcMap следует создать обработчик команды «Передать объект в АИС»:

```
Private Sub QueryByMap_Click()
Set pComServ = New MyComServ
fid = pFeature.Fields.FindField ("IDOBJECT")
Value = pFeature.Value(fld)
pComSerc.QueryByMap(Value) 'Вызов функции сервера и передача параметра
End Sub
```

Доступ к данным в ArcGIS

С помощью встроенного инструментария VBA, предназначенного для создания приложений в среде ArcGIS, можно получить доступ к любым свойствам слоёв и объектов на карте, а также создавать собственные процедуры для расчета определенных показателей.

Хотя инструментарий VBA в ArcGIS не позволяет перемещать объекты на карте, изменяя их позицию относительно слоя или в определенной системе координат, тем не менее, имея доступ к свойствам объекта, существует возможность моделирования состояний объектов на карте. Например, моделирование транспортной системы возможно реализовать с использованием модели клеточных автоматов.

ПРИМЕР. Доступ к карте и объектам на ней осуществляется с помощью следующих переменных.

```
Public Sub CommandButton1_Click()
Dim pDoc As IMxDocument
Dim pMap As IMap
Dim pLayer As ILayer
Dim pFLayer As IFeatureLayer
Dim pFClass As IFeatureClass
Dim LayerName As String
```

Присвоение переменным связей с картой осуществляется следующим образом.

```
Set pDoc = ThisDocument
Set pMap = pDoc.FocusMap
Set pLayer = pMap.Layer(0)
Set pFLayer = pLayer
Set pFClass = pFLayer.FeatureClass
Dim pFields As IFields
Set pFields = pFClass.Fields
```

Перебор всех точек на карте и назначение начальных величин осуществляется следующим образом.

```
Dim a (1 To 29) As Integer
Dim n As Integer
Dim x As Integer
Dim k1 As Integer
Dim pFeature As IFeature
Dim pFCursor As IFeatureCursor
Set pFCursor = pFClass.Update(Nothing, False)
Set pFeature = pFCursor.NextFeature
For n = 0 To 28
a (n + 1) = pFeature.Value(intposName)
Set pFeature = pFCursor.NextFeature
Next
```

После изменения состояний обязательно необходимо вызвать функцию **Query**.

```
Call Query3
```

Эта функция выделит все объекты на карте снова с учетом их новых характеристик. Описание объектов в функции следующее.

```
Public Sub Query3()
Dim pMxDoc1 As IMxDocument
Dim pMap1 As IMap
Dim pLayer1 As ILayer
```

```
Dim pFeatureSelection1 As IFeatureSelection
Dim pQueryFilter1 As IQueryFilter
Dim pFLayer1 As IFeatureLayer
Dim pActiveView1 As IActiveView
Dim pFClass1 As IFeatureClass
Dim pFeatureCursor1 As IFeatureCursor
```

Создание классов и объектов в функции следующее.

```
Set pMxDoc1 = ThisDocument
Set pMap1 = pMxDoc1.FocusMap
Set pLayer1 = pMap1.Layer(0)
Set pFLayer1 = pLayer1
Set pFeatureSelection1 = pFLayer1
Set pActiveView1 = pMxDoc1.FocusMap
If pFLayer1.FeatureClass.ShapeType = esriGeometryPoint Then
   Set pFClass1 = pFLayer1.FeatureClass
   Set pQueryFilter1 = New QueryFilter
   pQueryFilter1.WhereClause = "Objectid > 0 and ch = 1"
   pFeatureSelection1.SelectFeatures pQueryFilter1, _
        esriSelectionResultNew, False
   pActiveView1.PartialRefresh esriViewGeoSelection, _
        Nothing, Nothing
  Else
   MsgBox pLayer.Name & "   Это не точечный слой"
End If
End Sub
```

Вопросы для самоконтроля

1. Какие основные типы данных можно выделить в современных информационных системах?
2. Какие возможности предоставляет разработчикам ArcGIS для интеграции пространственной и атрибутивной информации?
3. Плюсы и минусы подходов по интеграции пространственной и атрибутивной информации и в каких случаях имеет смысл их использовать?
4. Каковы основные принципы связи пространственной и атрибутивной информации?

5. С помощью какого инструментария можно получить доступ к свойствам объектов карты в среде ArcGIS?

2.7 УПРАВЛЕНИЕ АТРИБУТИВНЫМИ ДАННЫМИ

Под управлением данными следует понимать выбор типов данных и подходящих структур их организации. Основу программного обеспечения в управлении пространственными данными составляет банк данных (БД) с соответствующей системой управления базой данных (СУБД), чья модель может быть иерархической, сетевой, реляционной, объектно-реляционной или объектно-ориентированной.

База данных – совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (СУБД).

Система управления базами данных – комплекс программных и лингвистических средств общего или специального назначения, реализующий поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей в условиях принятой технологии обработки данных. Если тематической информацией можно эффективно управлять в любой, в том числе наиболее распространенной – реляционной модели СУБД, то для геометрической, и как ее часть топологической, информации реляционная модель не эффективна. Для управления такой информации гораздо эффективнее сетевые модели данных. В настоящее время проводятся исследования по сочетанию этих преимуществ в базах данных основанных на объектной идеологии – в объектно-ориентированных БД. В связи с ростом объемов сбора и интеграции растровых данных в ГИС возникает задача общего управления такими данными.

Современные направления в исследовании управления и обработки данных следующие:

- определение оптимальных структур данных при гибридном управлении данными;
- объектно-ориентированные подходы к хранению информации, и к приемам работы с пространственными данными;
- языки описания и считывания пространственных данных.

Анализ данных в ГИС

Под анализом данных в ГИС понимают комплексный анализ данных ГИС (геометрических, топологических, тематических) с целью получения результата полезного для принятия решений.

Методы анализа данных в ГИС содержат геометрические, логические операции, операции реляционной алгебры, статистические операций. При этом существенное значение приобретает производительность этих методов, а также способ представления результатов пользователю.

Основа блока анализа – банк методов, где алгоритмы упорядочены по геометрии и тематике. Важными направлениями в области анализа данных ГИС являются:

- разработка новых методов и алгоритмов обработки пространственных данных (математическое описание).
- разработка программного обеспечения, реализующего основные методы комплексной обработки данных ГИС (геометрических, топологических и тематических данных).

Представление данных

Под представлением данных ГИС следует понимать методы, модели и способы визуализации пространственной информации. Компонента представления ГИС – самый важный аспект системы после анализа, поскольку визуализация результатов различными методами (картографии, рендеринга трехмерных сцен) имеет большие преимущества перед воспроизведенными таблицами цифр, что значительно повышает эффективность ГИС, восприятие информации пользователем.

Определение атрибутивных и пространственных запросов

Привязку пространственного объекта к его атрибутивному описанию следует осуществлять следующим образом.

В ArcMap следует создать процедуру.

Public Sub BindRequest (Number **As** String)

Цель создания процедуры – получить номер из АИС, который будет присвоен пространственному объекту.

Если необходимо создать пользовательский инструмент (UIToolControl) **BindTool** с обработчиком события **MouseDown.**

Private Sub BindTool_MouseDown (ByVal button **As** Long, _
ByVal shift **As** Long, ByVal x **As** Long, ByVal y **As** Long)

В АИС пользователь выбирает объект и сообщает системе о своем намерении выполнить привязку (нажимает кнопку или выбирает пункт меню). АИС вызывает функцию **BindRequest**, которой передает идентификатор привязываемого объекта. Идентификатор сохраняется в глобальной переменной VBA, и автоматически вызывается инструмент **BindTool**.

Когда пользователь щелкнет мышкой на карте (при нажатом инструменте **BindTool**), вызовется **BindTool_MouseDown**, цель которой найти объект, на котором произошел щелчок мыши, и присвоить его атрибуту (IDObject) значение, переданное из АИС.

Поиск объекта следует осуществлять с помощью обычного пространственного запроса по точке, а присвоение атрибута – с помощью интерфейса **IEditor**.

```
pID = "esriCore.Editor"
SetpEditor = Application.FindExtensionByCLSID(pID) 'Интерфейс IEditor
pEditor. StartOperation
pRow.Value(indlDOBJECT) = GlldObj 'Присвоение значения глобальной переменной
pRow. Store
pEditor.StopOperation "Привязка, Идентификатор = " + Str(GlIdObj)
```

Использование **IEditor** позволяет поместить каждую операцию привязки в стек операций редактирования и создать соответствующую команду отмены операции (**Undo**) в стандартном меню **Edit**.

ArcObjects предоставляет разработчику высокоразвитые средства для реализации интеграции пространственных и атрибутивных данных. Технология COM, лежащая в основе ArcObjects, позволяет разрабатывать компоненты информационной системы группой разработчиков с использованием любых средств разработки, поддерживающих COM.

Благодаря этому, ArcGIS можно рассматривать не только как мощную геоинформационную систему, но и как удобное средство разработки.

В основе векторного метода формализации пространственных данных лежит точка (point) – первичный графический элемент с координатами (X, У). Местоположение точки известно с произвольно заданной точностью. Две точки с координатами (Xl, Yl) и (X2, У2) формируют ли-

нию (line) – отрезок прямой, соединяющий эти точки, а замкнутая последовательность линий – полигон (polygon).

Совокупность этих элементов вполне достаточна для описания формы как линейных, так и площадных картографических объектов. Они кодируются как совокупность координат точек, аппроксимирующих форму линейного объекта. Например, аппроксимация административной границы, русла реки или контура (границы) территориального объекта, территории землепользования, населенного пункта, бассейна реки.

Вопросы для самоконтроля

1. Поясните принципиальное отличие базы данных от системы управления базами данных.
2. Что понимают под анализом данных в ГИС?
3. Какие основные направления в области анализа данных в ГИС?
4. Что следует понимать под представлением данных в ГИС?
5. Какие операции содержат методы анализа данных в ГИС?

ГЛОССАРИЙ

А

Автоматическое форматирование — Компонент, обеспечивающий автоматическое форматирование кода по мере ввода. При этом автоматически заменяются на прописные первые буквы ключевых слов, устанавливаются стандартные интервалы, добавляются знаки препинания, а также устанавливаются основной и фоновый цвета

Аргумент — Константа, переменная или выражение, передаваемые в процедуру

Б

Базовый класс — Исходный класс, от которого наследуются производные классы

Библиотека динамической компоновки (DLL) — Библиотека процедур, которая загружается в приложения и привязывается к ним во время выполнения. Для разработки библиотек DLL используются другие языки программирования, например, C, MASM или FORTRAN

Библиотека объектов — Файл с расширением OLB, в котором хранятся сведения о доступных объектах, предоставляемые автоматическими контроллерам, таким как Visual Basic. Для просмотра содержимого библиотеки объектов и сведений о хранящихся в ней объектов вы можете использовать обозреватель объектов

В

Время выполнения — Время выполнения кода. Во время выполнения код недоступен для редактирования

Время компиляции — Период, в течение которого осуществляется преобразование исходного кода в исполняемый

Время разработки — Этап, на котором осуществляется построение приложения в среде разработки, добавление в него элементов управления, настройка свойств и элементов управления форм и т. д. В отличие от этого этапа, во время выполнения можно взаимодействовать с приложением как его пользователь

Вставляемый объект — Объект приложения, который имеет тип настраиваемого элемента управления, например, лист Microsoft Excel

Встроенные константы — Константы, предоставляемые приложением. Константы Visual Basic перечислены в библиотеке объектов. Для их просмотра следует использовать обозреватель объектов. Встроенные константы нельзя отключить, в связи с чем не возможно создавать пользовательские константы с тем же именем

Выражение Сочетание ключевых слов, операторов, переменных и констант, позволяющее получить строку, число или объект. С помощью выражений вы можете выполнять вычисления, работать со строками и знаками, а также проверять данные

Д

Динамический обмен данными (DDE) Стандартный протокол для обмена данными с использованием активных каналов между приложениями под управлением ОС Microsoft Windows

Документ Любой изолированный объект, созданный с помощью приложения и обладающий уникальным именем файла

З

Знак объединения строк Сочетание знаков пробела и подчеркивания (_); используется в среде разработки для размещения одной логической строки кода на нескольких физически строках. Обратите внимание, что этот знак нельзя использовать для размещения таким образом строки кода со строковым выражением

Значок Графическое представление объекта или понятия. Чаще всего представляет свернутые приложения в Microsoft Windows. В качестве значка используются точечные рисунки размером не более 32 на 32 точек, хранящиеся в файлах с расширением ICO

И

Идентификатор Элемент выражения, задающий ссылку на константу или переменную

Исполняемый файл Приложение для ОС Windows, которое может выполняться вне среды разработки. Исполняемые файлы имеют расширение EXE

К

Класс Формальное определение объекта. По сути, класс представляет собой шаблон, на основе которого во время выполнения создается экземпляр объекта. В классе определяются свойства объекта и методы, управляющие его поведением

Код знака Номер знака в наборе символов, например, в наборе ANSI

Коллекция Объект, который содержит набор связанных объектов. При изменении коллекции позиция любого объекта в ней может изменяться. Таким образом, каждый объект коллекции имеет переменную позицию

Комментарий Текст в коде, поясняющий его работу. В Visual Basic строка комментария может начинаться апострофом (') или ключевым словом Rem с пробелом

Константа Именованный элемент, в котором во время выполнения программы хранится постоянное значение. Константа может содержать строковый или числовой литерал, другую константу и любое сочетание арифметических и логических операторов, за исключением Is и возведения в степень. В каждом ведущем приложении может определяться собственный набор констант. Пользователи могут определять дополнительные константы с помощью оператора Const. Можно использовать константы в любом месте кода вместо фактических значений

Конструктор Окно среды разработки Visual Basic, предназначенное для визуальной разработки. В этом окне представлены визуальные функции создания новых классов. В Visual Basic реализованы встроенные конструкторы форм. В версиях Visual Basic Professional и Enterprise представлены конструкторы элементов и документов ActiveX

Контейнер Объект, который может содержать другие объекты

Л

Логическая ошибка Ошибка программирования, в результате которой код перестает выполняться или возвращает неверные результаты. Например, причиной логической ошибки может быть неверное имя или тип переменной, бесконечный цикл, ошибки сравнения или неверное определение массива

Логическое выражение Выражение, при вычислении которого получается значение True (истина) или False (ложь)

М

Массив Набор последовательно индексируемых элементов, имеющих одинаковый внутренний тип данных. Каждому элементу массива присваивается уникальный индекс, идентифицирующий его в последовательности. Изменения одного элемента массива не затрагивают остальные его элементы

Массив элементов управления Группа элементов управления с общими именем, типом и процедурами событий. Каждому элементу в массиве присваивается уникальный индекс, который позволяет определить элемент, распознающий событие

Метафайл Файл, в котором хранится изображение, представленное не в виде точек, а в виде графических объектов, например, линий, окружностей и многоугольников. Метафайлы бывают двух типов: стандартные и расширенные. Стандартные обычно имеют расширение WMF, а расширенные — EMF. При масштабировании изображения в метафайле достигается более высокая точность

Метка строки — Идентифицирует отдельную строку кода. Метка строки может содержать любое сочетание знаков, но должна начинаться с буквы и заканчиваться двоеточием (:). Метки строк задаются без учета регистра и должны начинаться в первом столбце

Метод — Процедура, выполняющая действия в отношении объекта

Модуль — Набор объявлений и процедур

Модуль класса — Модуль, в котором содержится определение класса, а также его свойств и методов

Модуль кода — Модуль, в котором содержится открытый код, доступный всем модулям в проекте. В более поздних версиях Visual Basic модуль кода называется стандартным модулем

Модуль объекта — Модуль, содержащий код объекта. В качестве примера можно привести модули классов, форм и документов. Модули объектов содержат код программной части связанных с ними объектов. Правила работы с модулями объектов отличаются от стандартных модулей

Модуль формы — Файл с расширением FRM в проекте Visual Basic, который содержит графическое описание формы, ее элементы управления и их свойства; объявления констант, переменных и внешних процедур на уровне формы; а также процедуры событий и общие процедуры

H

Надстройка — Настраиваемое средство, позволяющее расширить возможности среды разработки Visual Basic

Номер ошибки — Целое число в диапазоне от 0 до 65 535, которое соответствует значению свойства Number для объекта Err. Номер ошибки в сочетании со значением свойства Description для объекта Err представляют полное сообщение о конкретной ошибке

Номер строки — Идентифицирует отдельную строку кода. Может содержать любое сочетание цифр, уникальное в рамках модуля, в котором оно используется. Номера строк должны начинаться с первого столбца

O

Область кода — Область окна кода, в которой вы можете вводить и редактировать код. Окно кода может содержать несколько областей кода

Общая переменная — Переменная, которая объявлена с помощью оператора Public и видима всем процедурам и модулям во всех приложениях, для которых не действует оператор Option

	Private Module. В последнем случае переменные будут являться общими только в рамках проекта, в котором они размещаются
Общая процедура	Процедура, которая должна явно вызываться из другой процедуры. В отличие от нее, процедура события вызывается автоматически в ответ на действие пользователя или системы
Объект ActiveX	Объект, взаимодействующий с другими приложениями или инструментами программирования через интерфейсы автоматизации
Объект источника событий	Объект-источник событий, возникающих в ответ на какое-либо действие. Объект источника событий возвращается свойством. Например, свойство CommandBarEvents возвращает объект CommandBarEvents.
Объявление	Неисполняемый код, который объявляет имя константы, переменной или процедуры, а также их характеристики, например, тип данных. В процедурах DLL с помощью объявлений задаются имена, библиотеки и аргументы
Оператор	Синтаксически полный элемент, который выражает один вид действия, объявления или определения. Оператор обычно занимает одну строку, однако при необходимости вы можете включить несколько операторов в одну строку, разделяя их двоеточием (**:**). Кроме того, можно разместить одну логическую строку на нескольких физических, используя знак объединения строк (_)
Оператор сравнения	Знак или символ, устанавливающий соотношение между несколькими значениями или выражениями. К ним относятся следующие: «меньше» (<), «меньше или равно» (<=), «больше» (>), «больше или равно» (>=), «не равно» (<>) и равно (=). Кроме того, к ним относятся операторы Is (является) и Like (подобно). Обратите внимание, что операторы Is и Like нельзя использовать в качестве операторов сравнения в операторе Select Case

П

Параметр	Имя переменной, которое используется внутри процедуры для обращения к переданному в процедуру аргументу. Этой переменной присваивается значение переданного в процедуру аргумента. Область видимости параметра ограничена областью процедуры
Переменная	Именованный сегмент памяти, который может содержать данные и изменяться во время выполнения программы.

	Каждой переменной присваивается уникальное в ее области видимости имя. Тип данных указывать не обязательно. Имена переменных должны начинаться с буквы, должны быть уникальными в заданной области видимости, и, кроме того, не могут содержать более 255 знаков, точки и символы объявления типа
Переменная модуля	Переменная, объявленная вне кода процедур Function, Sub или Property. Переменные модуля объявляются вне его процедур, существуют только пока модуль загружен и видимы для всех процедур модуля
Побитовое сравнение	Побитовое сравнение расположенных в одинаковых позициях битов для двух числовых выражений
Подпрограмма	Процедура, которая выполняет конкретную задачу в рамках программы, но не возвращает значения явно. Подпрограмма начинается и заканчивается операторами Sub и End Sub соответственно
Пользовательский тип	Любой тип данных, определенный с помощью оператора Type. Пользовательские типы могут содержать один или несколько элементов любого типа данных. С помощью оператора Dim можно создавать массивы пользовательских и любых других типов данных. Кроме того, пользовательские типы могут содержать массивы любых типов
Приложение	Набор элементов кода и интерфейса, выполняемых как единая программа. Для создания и выполнения приложений разработчики могут использовать среду разработки. Пользователи чаще всего запускают приложения в виде исполняемых файлов вне среды разработки
Проект	Набор модулей
Процедура	Именованная последовательность операторов, которая выполняется как единый блок. В качестве примера типов процедур можно привести Function, Property и Sub. Имя процедуры всегда определяется на уровне модуля. Весь исполняемый код должен быть заключен в процедуры. Вложение процедур друг в друга не допускается
Путь	Строковое выражение, задающее местоположение каталога или папки. При необходимости в пути может указываться буква диска

С

Свойство	Именованный атрибут объекта. Свойства определяют такие характеристики объекта, как размер, цвет, положение на экране или состояние (включен или отключен)

Среда разработки Компонент приложения, в котором пишется код, создаются элементы управления, настраиваете свойства и элементы управления форм и т. д. Противопоставляется выполнению приложения

Стек Фиксированный объем памяти, используемый средой Visual Basic для хранения локальных переменных и аргументов во время вызова процедуры

Т

Тип данных Характеристика переменной, определяющая тип содержащихся в ней данных. Поддерживаются следующие типы данных: Byte, Boolean, Integer, Long,Currency, Decimal, Single, Double, Date, String, Object, Variant (по умолчанию), а также определяемые пользователем типы и специальные типы объектов

Тип данных Boolean Логический тип данных с двумя единственными возможными значениями: True (-1) или False (0). Переменные типа Boolean хранятся в 16-битных (2-байтовых) числах

Тип данных Byte Тип данных, содержащий целые положительные числа в диапазоне от 0 до 255. Переменные типа хранятся в виде отдельных 8-битных (1-байтовых) чисел без знака

Тип данных Currency Тип данных со значениями в диапазоне от -922 337 203 685 477,5808 до 922 337 203 685 477,5807. Используется для операций с денежными средствами и вычислений с фиксированной запятой, где особо важна точность. В Visual Basic тип Currency представлен символом объявления типа @ (собака)

Тип данных Date Используется для хранения вещественного числа, представляющего дату и время. Переменные Date хранятся в виде 64-разрядных (8-битных) чисел. Значение слева от десятичного разделителя представляет дату, а справа — время

Тип данных Integer Тип данных, который содержит целочисленные переменные в виде 2-байтовых целых чисел в диапазоне от -32 768 до 32 767. Кроме того, тип данныхInteger используется для представления значений перечисления. В Visual Basic тип Integer представлен символом объявления типа % (процент)

Точка останова Строка программы, в которой автоматически останавливается выполнение. Точки останова не сохраняются вместе с кодом

Ф

Фокус — Способность элемента в конкретный момент времени считывать нажатия кнопок мыши или клавиш на клавиатуре. В среде Microsoft Windows в любой момент времени фокус может иметь только одно окно, одну форму или один элемент управления. Строка заголовка объекта, который находится в фокусе, обычно выделяется. Фокус может устанавливать как пользователь, так и само приложение

Форма — Окно или диалоговое окно. Формы представляют собой контейнеры для элементов управления. Формы многодокументного интерфейса (MDI) также могут выступать в качестве контейнеров для дочерних форм и некоторых элементов управления

Функция — Процедура, которая выполняет конкретную задачу в рамках программы и возвращает значение. Функция начинается и заканчивается операторами Function и End Function соответственно

Ч

Числовое выражение — Любое выражение, при вычислении которого получается число. Может содержать любое сочетание ключевых слов, переменных, констант и операторов, вычисление которых дает число

Э

Элемент — Элемент коллекции, объекта или определяемого пользователем типа

Элемент ActiveX — Размещаемый на форме объект, который расширяет возможности пользовательского интерфейса приложения. Элементы ActiveX поддерживают события и могут встраиваться в другие элементы управления. Файлы таких элементов имеют расширение OCX

Элемент управления — Размещаемый на форме объект, который обладает собственным набором распознаваемых свойств, методов и событий. Элементы управления могут принимать вводимые пользователем значения, отображать выходные значения и запускать процедуры событий. Для управления большинством таких элементов используются методы. Элементы управления могут быть интерактивными (реагируют на действия пользователя) и статическими (доступны только через код)

СПИСОК РЕКОМЕНДУЕМЫХ ИСТОЧНИКОВ

Основные

1. Програмування геоінформаційних задач. Лабораторний практикум: навч. посібник. – Харків : ХНУМГ ім. О. М. Бекетова, 2015. – 111 с.

2. Exploring Arcobjects. – ESRI, Redlands, 2004 г. – 450 с.

3. ArcTooIBox: Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2003. – 105 с.

4. ArcMap: Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2003. – 220 с.

5. ArcCatalog: Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2003. – 180 с.

Дополнительные

6. Попов И. В. Эффективное использование ArcObjects. Методическое руководство / И. В. Попов, М. А. Чикинев. – Новосибирск: СО РАН, 2003. – 160 с.

Ресурсы сети Internet

7. Сайт справки и обучения по приложениям Office корпорации Microsoft [Электронный ресурс]. – Режим доступа : http://office.microsoft.com/ru-ru/access-help – Загл. с экрана.

8. Сайт «ArcGIS resource» [Електронний ресурс]. – Режим доступа : http://www.dataplus.ru/index.php – Загл. с экрана.

9. Сайт додатків Office корпорації Microsoft [Електронний ресурс]. – Режим доступу : http://office.microsoft.com/ru-ru/access-help/.

10. Сайт корпорації «ArcGis Recource Center» по роботі з програмним продуктом – Режим доступу : http://help.arcgis.com/.

11. Сайт «Высокие технологии». – Режим доступа : http://www.citymap.odessa.ua.

12. Цифровий репозиторій ХНУМГ ім. О. М. Бекетова [Електронний ресурс]. – Режим доступу : http://eprints.kname.edu.ua.

Навчальне видання

ПОМОРЦЕВА Олена Євгенівна

«ПРОГРАМУВАННЯ ГЕОІНФОРМАЦІЙНИХ ЗАДАЧ»

НАВЧАЛЬНИЙ ПОСІБНИК

(Рос. мовою)

Відповідальний за випуск *К. А. Мамонов*

Редактор *О. В. Михаленко*

Комп'ютерний набір *О. Є. Поморцевої*

Комп'ютерне верстання *І. В. Волосожарової*

Дизайн обкладинки *Д. С. Мухіної*

Підп. до друку 23.12.2016 р. Формат 60×84/16
Друк на ризографі Ум. друк. арк. 5,0
Тираж 300 пр. Зам. №

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017 р.